Create Your Own Incredible 3-D Stereograms with Software Included!

Hidden Images:

Making Random Dot Stereograms

Discover how they work, how to view them, and how to customize your own.

Create beautiful and intriguing color field background patterns!

Design interesting objects which stereoscopically emerge before your eyes!

que

Hidden Images: Making Random Dot Stereograms

Software included!

que

Hidden Images: Making Random Dot Stereograms

Bob Hankinson

Alfonso Hermida

QUE

Hidden Images: Making Random Dot Stereograms

©1994 by Que®Corporation

All rights reserved. Printed in the United States of America. No part of this book may be used or reproduced in any form or by any means, or stored in a database or retrieval system, without prior written permission of the publisher except in the case of brief quotations embodied in critical articles and reviews. Making copies of any part of this book for any purpose other than your own personal use is a violation of United States copyright laws. For information, address Que Corporation, 201 W. 103rd St., Indianapolis, IN 46290.

Library of Congress Catalog No.: 94-68570

ISBN: 1-56529-994-9

This book is sold *as is,* without warranty of any kind, either express or implied, respecting the contents of this book, including but not limited to implied warranties for the book's quality, performance, merchantability, or fitness for any particular purpose. Neither Que Corporation nor its dealers or distributors shall be liable to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to have been caused directly or indirectly by this book.

97 96 95 94 6 5 4 3 2 1

Interpretation of the printing code: the rightmost double-digit number is the year of the book's printing; the rightmost single-digit number is the number of the book's printing. For example, a printing code of 94-1 shows that the first printing of the book occurred in 1994.

Publisher: David P. Ewing

Associate Publisher: Corinne Walls

Publishing Director: Brad R. Koch

Product Marketing Manager: Greg Wiegand

Dedication

To my wife, Karen.

—BH

To my parents and to my wife, Linda.

—AH

Credits

Publishing Manager
Brad R. Koch

Acquisitions Editor
Cheryl Willoughby

Product Director
Lisa A. Bucki

Production Editor
Danielle Bird

Editors
Kelli Brooks
Geneil Breeze

Technical Editor
Alfonso Hermida
Ron Holmes

Figure Specialist
Cari Ohm

Graphic Image Specialists
Dennis Sheehan
Jeff Yesh

Book Designers
Paula Carroll

Cover Designer
Jay Corpus

Production Team
Stephen Adams
Angela Bannan
Karen Dodson
Chad Dressler
DiMonique Ford
Beth Lewis
Malinda Lowder
Steph Mineart
G. Alan Palmore
Clair Schweinler
Kris Simmons
Kelli Widdifield
Donna Winters

Indexer
Johnna VanHoose

Editorial Assistant
Andrea Duvall

Acquisitions Assistant
Ruth Slates

Composed in Stone Serif and MCPdigital by
Que Corporation

About the Authors

Bob Hankinson

Bob Hankinson is an Electrical Engineer and former Technical Instructor with several years of digital design experience in various areas of video and graphics processing.

Bob is originally from Johnstown, Pennsylvania, and obtained his BS in electrical engineering from the Florida Institute of Technology. When they're not traveling around the globe, he and his wife Karen share a home with their dalmatians in a suburb of Dallas, Texas.

Alfonso Hermida

Alfonso Hermida has a BS and an MS in mechanical engineering from the University of Puerto Rico and an MS in mechanical engineering/space systems from George Washington University. He currently works at the NASA/Goddard Space Flight Center in Maryland as an aerospace engineer. He is also an adjunct professor at the Mathematics Department at Capitol College in Laurel, Maryland. He wrote Que's *Adventures in Ray Tracing* and is coauthoring a computer graphics book that will be published soon. Alfonso also has written several articles for the magazine, "3D Artist."

Acknowledgments

My whole interest in stereogram images began when my wife bought an early, primitive Random Dot Stereogram poster for me. Little did she know how much this simple gift would change our lives.

First and foremost, I want to thank my wife Karen. Tight schedules and deadlines took away a lot of time that I would have rather spent doing things with her. I thank her for her understanding, support and patience. Thank you, Sugar.

I also want to thank my friend Gary Peterson, a software engineer with a flair for excellence. His long hours, expertise, and professional attitude transformed a dull program into a high quality piece of software.

For their ideas on turning stereograms into a craft, I offer special thanks to my parents, Bob and Marion.

Finally, I want to thank Cheryl, Danielle, Brad, and the rest of the team at Que. Hard work, late nights, and top-notch professionalism sweetened with a good dose of humor helped make this a fun project.

Bob Hankinson

I would like to thank: Brad Koch, Cheryl Willoughby, Ruth Slates, and the rest of the Que team for all their help and for the opportunity to write part of this book; Bob Hankinson for his support and contributions; and my wife for being extremely patient with me.

Alfonso Hermida

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Corporation cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Overview

Table of Contents

Introduction

Stereograms are those funny looking images you see in many malls and stores. They come in a variety of different forms; some have been created with thousands of tiny, colored dots, while others seem to have been made using some sort of graphic design—even photographs. They've been printed on posters, postcards, T-shirts, and coffee mugs. They've even been included as promotional gifts with a number of different products. Supposedly, they've all been created in such a way as to yield a hidden, three-dimensional picture.

These images are incredibly complicated works of art. They have been created by large mainframe computers, requiring hours and hours of processing time. Only a handful of scientists and engineers possess the mathematical abilities necessary to create these images. The average person simply does not have the specialized education and training in this field. Even if he or she did, they almost certainly wouldn't have access to the special equipment required to generate this kind of image.

Don't you believe it! While these fascinating images seem to be very complicated, they are in reality very simple and easy to create. Furthermore, you have the ability to create these incredible images right on your IBM-compatible personal computer. This book not only shows you how, but also will furnishes you with the software!

What Sort of Information is Contained in This Book?

This book probably contains just about everything you ever wanted to know about stereograms. Chances are, it also contains a lot of things about stereograms that you never even thought about. Most importantly, it shows you how to create your own stereograms.

Some Basic Stereogram Theory

The word "theory" may sound a bit scary, but don't worry. A couple of chapters show you just how easy stereogram generation can be! Contrary to what many people believe, the creation of these images is really quite simple. Anyone can easily learn the basic concepts behind how this amazing optical illusion really works.

A Bit of the Nitty Gritty behind Stereograms

For those of you who want to probe a little deeper, this book also includes a few chapters dedicated to the down-and-dirty details of how a stereogram can be created. It includes discussions on some of the many different ingredients that can affect the quality and appearance of the images.

Some Really Good "How To" Topics

Nearly half of this book is dedicated to the subject of how to create stereograms. Some excellent, detailed, step-by-step discussions were written specifically for the reader who is interested in creating stereograms. You'll find out just how easy it is to create very high quality images on your personal computer. You'll be pleasantly surprised when you find out that you don't have to be a computer programmer to create stereograms. The only requirement is a desire to generate images of your own design.

Pictures and Software!

You can't write a book about stereograms without including pictures. This book's got pictures: real stereograms created by real people on plain old personal computers. A lot of sample stereogram images are spread throughout the book, including pictures that show the behind-the-scenes work in creating stereograms.

This book also provides a disk with an easy to use, award-winning stereogram creation program called POPOUT-LITE. This program runs under Microsoft Windows, the most widely used interface in personal computing. From simple drawings that *you* create using Microsoft Paintbrush, POPOUT-LITE creates personalized stereogram images!

Color Plates

Sixteen pages of color images are referred to from various places within the book. These reproductions add a sense of "realness" to both the theory and creation of stereogram images. All the color stereograms on these pages, as well as the many black-and-white stereograms throughout the book, were created using either the POPOUT-LITE that comes with this book, or its big brother POPOUT-PRO.

How is This Book Arranged?

This book is written to appeal both to the reader who wants to· understand some of the principles behind stereogram images, and the reader who wants to create images.

Part I: The World of Random Dot Stereograms

The first half of this book is dedicated to giving the reader an understanding of the basic concepts behind stereograms and how the POPOUT-LITE software actually creates the images. The goal of this section is to help the reader understand that while the images may look very complicated, they can actually be created quite easily!

Chapter 1: Introduction to Random Dot Stereograms

Chapter 1 is an introduction to the whole stereogram phenomenon. Here, you learn about terminology associated with stereograms, possible uses for your creations, and the roles of personal computers in creating these images.

Chapter 2: The Basic Concepts

In chapter 2, you gain a bit of insight on how your brain perceives depth and learn a new way to stare at a repeating pattern. You learn how easy it is to fool the brain, and how you can use the brain's tendency to make perceptual mistakes to create the illusion of depth on a simple, flat sheet of paper.

Chapter 3: Fundamentals of Creating a Stereogram

This chapter guides you through the actual creation of a stereogram. You learn about the essential elements that go into generating one of the images and see just how easy the process can be.

Chapter 4: Understanding Stereogram Algorithms

For those of you with a flair for computer programming, this chapter shows how the concepts of chapter 3 can be broken down into simple, easy-to-follow steps. For the non-computer programmer, chapter 4 offers a helpful insight into the behind-the-scenes details of what steps POPOUT-LITE follows to convert a Paintbrush drawing to a stereogram.

Chapter 5: Stereogram Generation Parameters

Chapter 5 is provided as a helpful reference that explains some of the essential ingredients that affect the overall viewability, quality, and appearance of the final stereogram. Once you've begun to create stereograms, you will find yourself wanting to improve the quality of the images.

Chapter 6: Advanced Topics

This chapter is dedicated to some discussions and explanations of the more obscure adjustments and recipes that are a part of stereogram creation. You are also introduced to another, even more impressive, type of stereogram image!

Part II: Generating Random Dot Stereograms

Now that you understand the principles behind the images, Part II guides you through the actual process used to create stereograms! This half of the book includes a number of tips and hints that will help you create high quality images in no time!

Chapter 7: Creating a Depth Image with Paintbrush

POPOUT-LITE was created especially for use with Windows' Paintbrush program. Chapter 7 gives you some background on this popular drawing utility. Even those who have experience with Paintbrush will find this chapter useful, as it is tailored to the generation of stereogram depth images.

Chapter 8: Drawing Complex Objects

Chapter 8 walks you through the steps used in creating various kinds of objects for use in the generation of stereograms. You learn some handy tips that can help to make depth image creation a little easier.

Chapter 9: Using POPOUT-LITE

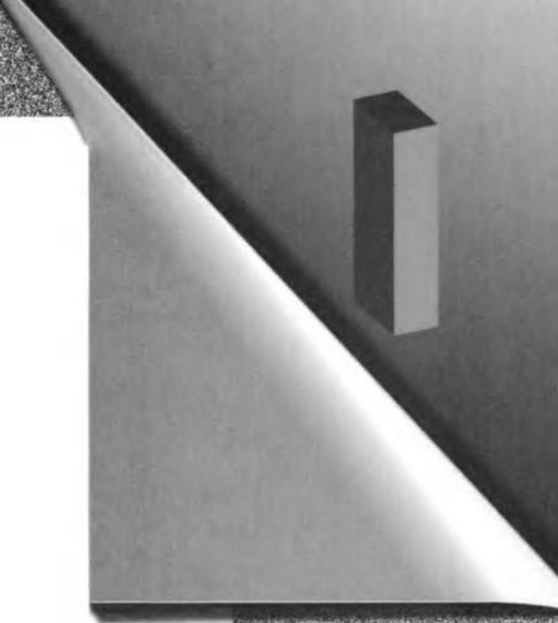
Once you've created your depth file using Paintbrush, chapter 9 walks you through the few simple steps for turning it into a random dot stereogram. Chapter 9, along with POPOUT-LITE's extensive built-in online Help File, should answer just about any question you've got about the software.

Chapter 10: Converting and Using Random Dot Stereograms

With chapter 10, we present some final ideas that may help you in creating your own stereograms and include some possible uses for your stereogram images. You are introduced to some of the alternative ways of creating depth images and learn where to obtain depth images that have been created by others. You also learn how to make the images that you've created available to other people with similar interests.

Get Ready...

Fire up that computer of yours and get those disk drives spinning, because we're about to introduce you to one of the most fun and amazing things you can do with your computer. We take all the magic out of stereograms, and show you exactly what it takes to create these truly remarkable three-dimensional works of art!



Part I: The World of Random Dot Stereograms

1

Introduction to Random Dot Stereograms

"WOW!" pretty much sums up the topic of stereograms. If you've seen them, chances are you'd sum them up in much the same way. Of course, chances are also that if this whole book consisted only of the word, "WOW," you'd think you got a pretty lousy deal. What we're going to do, then, is try and expand "WOW" into information that's a bit more useful and practical. If you've seen Random Dot Stereograms, you understand that they're pretty intriguing. If you haven't, you'll be saying, "WOW" in no time.

After providing some background information about what random dot stereograms are, this book takes you through a step-by-step understanding of how these amazing optical illusions work and offers some insight as to how the images are actually created. In addition, the book gives you the practical knowledge necessary to create your own personalized stereograms.

Hidden Images: Making Random Dot Stereograms also includes actual, working demonstration versions of some high quality stereogram generation software for your computer. After reading this book, you'll be able to create your own stereograms, and you'll have a very good understanding of what goes into creating one of these incredible illusions.

What are Random Dot Stereograms?

There's a pretty good chance that if you're reading this book, you already have some idea of what the term *random dot stereogram* refers to. We've all seen them, or at least we've seen the groups of people standing around a mall kiosk to get a better look at one of them. Stereograms have shown up in malls and airports, on T-shirts, coffee mugs, postcards—you name it. The phenomenon has swept the United States, and quite a few other countries as well.

Random dot stereograms are pictures, or images, that at first appear to be nothing more than a series of randomly colored dots or patterns. Some of them are quite colorful and attractive, reminiscent of some kind of "modern art" from a paisley past. When you first see one, you may find yourself asking a question such as, "How do you know which way is *up*?"

The fun part comes when you're looking at one of these things, wondering what in the world motivated someone to create such a mess, and then you hear someone exclaim something like, "WOW." The old fable about the Emperor's new clothes quickly comes to mind as more and more people gather and stare, and then proceed to yell things like, "I see it!" It doesn't take long to realize that there seems to be some incredibly obscure but important difference between seeing a stereogram and *seeing* a stereogram!

I met a woman a while back who was convinced that stereograms were simply the 90s version of "pet rocks." There's really nothing to them," she insisted. "People want to believe that when they look at the pictures, they see something that's really not there. If they believe they're supposed to see something, their minds will convince them that there really is something there to see, just like seeing objects within clouds. It's obvious that the only thing in the picture is a bunch of pretty patterns."

Do you agree? If you do, this section has been written specifically with you in mind because I'll guarantee you, a stereogram ain't no pet rock!

Let's begin with the phrase *random dot stereogram.* Technically, there are a couple of different types of images like this. A more general term for this type of image is *stereoscopic image,* or *autostereogram.* A few other terms are used to describe the image, but the most common and descriptive term, and the one used in this book, is *stereogram.*

The first modern stereograms that were popularized in recent years were created using a whole bunch of randomly colored dots. From this, the name *random dot stereogram* became a commonly used phrase to describe the images. The term is often abbreviated as *RDS*. Technically, the common RDS image is a special case of a more general type of optical illusion created using random dots. This special case image is referred to as a *single image random dot stereogram,* or *SIRDS.*

NOTE: In theory, multiple hidden images can be placed within one stereogram. A stereogram that contains two different images is referred to as a *dual image random dot stereogram.*

Common stereograms are usually referred to as *RDS images*. Technically, this is incorrect, as not all stereograms are created using random dots. In chapter 3, for example, you will see how stereograms can also be created using a nonrandom pattern, or even an image produced from scanning a photograph.

A stereogram is an optical illusion. Compared to other optical illusions you may have seen, stereogram is extremely complex. When you first look at one, you see just a series of random looking dots or patterns. When you stare at a stereogram in just the right way, however, you see a previously hidden, second image suddenly appear! Moreover, you see this image in three dimensions.

This isn't just a psychological phenomenon as suggested by the woman I mentioned earlier. The random dots or patterns that make up the stereogram are not really random at all. They've all been carefully and precisely calculated by a computer to create this amazing optical illusion. Rest assured that when you hear the exclamation, "I see it!," the person saying it has just viewed the stereogram in the "proper" manner, and now sees this "hidden," three-dimensional image.

The following chapters explain the principle behind this "proper" manner of viewing a stereogram. You'll learn how this proper viewing, combined with the precise way in which the image was created, yields a true, three-dimensional image from a seemingly random picture.

History of Stereoscopic Images

Before proceeding any further, I want to briefly touch on the concept of three-dimensional optical illusions in general. At the most fundamental level, there's really nothing extra special about stereograms. After all, folks have been viewing three-dimensional optical illusions for years! Consider the case of the old stereo viewers of the 19th century. These novelties consisted of two almost identical photographs, mounted side by side. This double photograph was placed behind two viewing lenses. The observer looked through the viewing lenses in much the same way that one looks through a pair of sunglasses. The double photo was mounted so that the observer's right eye could only see the photograph on the right, while the left eye saw only the image on the left. The result was that the observer saw only a single image that appeared to be three dimensional!

A more modern version of this same concept has evolved into the handheld viewers that many children possess. These have also been around for years. Slip a cardboard disk into the viewer, point it at a source of light, and proceed to view a whole series of three dimensional images just by moving the slider mechanism.

Of course, the major difference between these types of illusions and stereograms is that they involve using two different images, whereas stereograms are single images. With two images, the three-dimensional effect is created by taking two photographs instead of just one. The two photos are taken with a camera that has two lenses, spaced about the same distance apart as the distance between your eyes.

You "see" a three-dimensional image instead of just a flat photograph because your two eyes are each looking at a slightly different view of the same scene. You normally view your surroundings in three dimensions because your two eyes see a slightly different view of the world in the first place!

With a stereogram, these two different views are combined and pressed into a single image. The "trick" is to view the stereogram in such a way as to effectively "separate" this single image into two different images. Because of the way a stereogram is created, it can be viewed in such a way that forces your two eyes into seeing slightly different views of the same thing. The result, determined by exactly how the stereogram was generated, yields a three-dimensional object or scene. When the eyes are "tricked" into seeing slightly different views of the same object, the illusion of depth can be created, just like with the old stereo viewers. The major difference is that with a stereogram, both of these slightly different views are buried within the same image, rather than being two different images. In chapter 3, you will see exactly how this can be accomplished.

This concept of combining two views into a single image is really not all that new. I remember seeing an optical illusion years ago that consisted of rows of pictures of the sun's planets. Within a row, each instance of a planet was slightly different than the planet next to it. When you stared at the image in just the right way, the different planets would appear to be at different distances from you, and appeared to be actual spheres. The rings of Saturn even looked three-dimensional.

The "father" of the stereogram-type of illusion is actually just as old as the stereo viewers of the 19th century. It wasn't at all popular, simply because the illusions that were produced were very simple. In chapter 2, I've included a figure where an observer is looking at a row of objects, and each object is slightly different than the one next to it. This type of "repeating pattern" illusion was introduced back in the 19th century, and contains the basic elements of modern stereograms. Since that time, there have been numerous papers written on the subject, and numerous algorithms developed for creating such images. Also, the stereograms of today are much more complex, and contain much more detail.

The stereogram images of today can trace their conceptual roots back to the optical illusions that consist of a row of slightly different objects. The biggest difference is only in the detail and resolution of the modern images. Today's images are created using computers, which allow for processing the huge amounts of individual dots that make up modern stereograms.

It is important to realize that the fundamental concepts and principles behind how stereograms work go back to a pre-computer era. This means that computers are really not necessary for creating stereograms, and you really don't need to know anything about computers in order to understand the concepts behind the illusion. Of course, computers really are used to create stereograms. In reading the following chapters, you will find that while the steps to creating the illusion are quite easy, they are also very tedious and repetitive.

Stereograms and Computers

As I just mentioned, computers are really not an essential ingredient in the creation of stereogram images. You can fully understand the fundamentals of how the images work and at the same time, know nothing about computers.

Stereograms are a collection of dots. That's all. You can make dots with a pencil on a piece of paper, with a hammer, punch, and a piece of wood, or with needlepoint. Bathroom tiles are really just huge square dots, and push-pins are like big round dots. Actually, you can make dots in a whole lot of different, creative ways. You can create stereogram images on just about any medium where you can make dots! Once you understand the principle behind stereograms, you don't need a computer at all.

Now, if you were to take a close look at the average computer-generated stereogram, you'd see that what I say is true; that a stereogram is really a collection of dots. If you really get down to it, modern computer-generated stereograms are just an extremely large collection of extremely little dots!

Two necessary ingredients in creating a stereogram are a lot of patience and perseverance. While the amounts of these ingredients vary in different individuals, we can all agree that we have limited amounts. Computers, on the other hand, are really stupid machines that were created to have infinite amounts of both patience and perseverance. This, plus the fact that they can do arithmetic much faster than you or me, makes them an ideal tool in stereogram creation.

Modern stereograms are all created by computer, simply because of the massive numbers of dots used to create the image. There's no "higher math" operations that go into creating stereograms. Theoretically, anyone can do them with a little time and patience.

I'm still trying to convince Mom to knit a stereogram into one of her afghans. So far, no luck. I get a similar reaction from Dad when he realizes how many holes I want him to punch in that piece of wood. I just wish he'd quit telling her, "That boy needs help, Marion."

My wife has a whole mess of push-pins that she uses on a big map at work to keep track of her business trips. I've tried to convince her that I know of a much better way that she could use those push-pins, and at the same time help to occupy her time while on the road. Suffice to say that when my wife stares, it isn't always at a stereogram.

The bottom line in all of this is that, practically speaking, you're probably much better off using a computer to create stereograms. With the availability of PCs in today's world, and the emergence of quality software for creating stereograms, there's really no good reason to do something like this manually.

Even the persevering individual who wants to create a stereogram using arts and crafts would find a computer to be very helpful in the process. Manually created stereograms can use the computer to actually output the image, which then becomes a sort of "template" for the actual crafting of the final product. For the vast majority, the PC-generated stereogram will be the final product of our efforts. Of course, this not only includes viewing the image on a PC monitor, but also printing the stereogram to show others.

Stereogram creation software for the PC offers many an introduction to what has been a very exclusive club. No one knows how many potential stereogram artists are out there. People who can't tell one end of a paintbrush from the other may very well find their special artistic niche to be in the creation of stereograms.

The Software that Comes with This Book

When you read this book, you'll notice that a lot of the concepts apply to many good stereogram programs that are available for the PC. We also realize that you may not already possess stereogram software, so we've included an award-winning piece of software called POPOUT-LITE.

POPOUT-LITE is an easy to use stereogram generator that has received numerous favorable reviews from PC magazines. POPOUT-LITE operates under Microsoft Windows, and will allow you to generate stereograms from drawings you create using Microsoft's Paintbrush program. While POPOUT-LITE is a fully functional random dot stereogram generator, some readers may find that they want to create even more sophisticated stereograms. We've included a discount coupon with the book allowing you a special reader's discount to POPOUT-PRO version 2.2, the easiest to use and most powerful stereogram shareware in its class!

Creating a Simple Stereogram with POPOUT-LITE

POPOUT-LITE was designed to be easy to use. If you're not sure how to do something, click Help in the POPOUT-LITE main menu. We've tried to include a lot of tips and hints throughout the help file.

POPOUT-LITE accepts 16-color BMP format images as depth files. These are easy to create with Paintbrush, and we show you just how to do it in Part 2 of this book. For your first stereogram, you may want to use one of the sample depth files that are included with POPOUT-LITE.

The following steps show you how to create a stereogram from a depth file that we've included on the disk:

1. Install and start POPOUT-LITE (See the last page of this book for installation instructions.)
2. In the POPOUT-LITE main menu, click the Depth File button. Select the file EXAMPI1.BMP from the directory where you installed POPOUT-LITE. This is a depth image in 16-color BMP format. To view this image, click View, and then click Depth File

from the POPOUT-LITE main menu. This image is hidden in the stereogram. Each of the colors of this image creates a different level of apparent depth in the stereogram.

3. From the POPOUT-LITE main menu, choose Stereogram File. Make up a name like MYTEST.BMP. Make sure to include .BMP.
4. Next, click the Generate button in the POPOUT-LITE main menu. This creates a stereogram called MYTEST.BMP that contains EXAMPI1.BMP as the hidden image.
5. When the image is finished, click View, and then click Stereogram File to view the stereogram you've just created. Can you see it? That's all there is to it!

POPOUT-LITE will create random dot stereograms containing up to 16 levels of apparent depth. You can create your own depth images using Paintbrush—just remember to save them as 16-color BMP files. In Paintbrush, you can redefine the palette by clicking Options, Get Colors. If you select the file POPOUT.PAL, your Paintbrush palette changes to show you the 16 colors that POPOUT-LITE recognizes. Simply use these 16 colors to draw your own depth images. The ordering of the colors in the Paintbrush palette matches the levels that will be created. Black will be the level furthest from you, and White will be the level closest to you.

Don't be afraid to experiment with some of the different options of the program. You can create randomly colored RDS images, or select the set of colors used like the images of color plate 2 and color plate 7, or you can create custom colored stereograms like those of color plates 5 and 6. You can also experiment with Pattern Width and Pixel options.

If you're not sure about something, refer to the Help file. Just click Help in the POPOUT-LITE main menu, and then click Contents. This takes you to the main menu of the Help File. From there, you can click any of the displayed topics, or click the Search button to see a list of all available Help Topics.

Stereogram Terminology

One thing that is consistent in stereogram generation is the inconsistency of some of the terminology used! Unfortunately, there's no single "Stereogram Dictionary" that one can refer to in order to find the "proper" terms.

What we've done is try to compile an admittedly limited list of some of the more common terminology that is used throughout this book and elsewhere. The idea is to show you that while there may be quite a few different terms used on the subject of stereograms, many of those terms all refer to the same thing! Hopefully, this will give you a central reference point if you come across a term that you don't quite understand, or you're not sure what it refers to.

For example, *stereogram* is just one of many terms used to describe the images that this whole book is written about. The following terms are used in reference to stereograms:

Stereogram
Random dot stereogram
RDS
SIRDS
Single image random dot stereogram
Autostereogram
Stereoscopic image
Holusion
RDS hologram
Dot image

All stereograms are created using some sort of starting image. This is the image that will be hidden in the stereogram. Some of the different terms to describe this image are:

Depth file
Depth image
Depth map
Depth field
Height file
Height image
Height map
Height field
Source file
Source image
Hidden image
Destination image

All stereograms discussed in this book are created using some sort of horizontally repeating pattern. Some of these patterns are random, while others are not. This pattern has names such as:

Random field
Random pattern
Psuedorandom pattern
Substitution pattern
Output pattern
Pattern
Repeating pattern
Custom generated random field
Colorfield file
Colorfield image
Tile
Strip
Background image
Repeating sequence

Summary

This book isn't filled with a lot of equations and mathematical concepts, nor is it intended to be some sort of high level intellectual paper full of theories and abstractions. On the contrary, it tries to maintain a common sense, common person approach to the topic of how stereograms are created. Don't misinterpret this book as being some all-knowing authority on the subject of stereograms. After reading this book, you should be able to look at a stereogram and say, "I really understand the basic concepts of how this is created." If you try some of the examples that are presented, you'll find that anyone can create personalized stereograms.

2

The Basic Concepts

Stereogram images are truly incredible! Not only do they have an uncanny ability to attract crowds, but if you've ever been fortunate enough to properly view a well-crafted stereogram, you'll agree that they really are amazing.

If you've ever been near a crowd of people standing around a collection of stereograms, I'll bet you've also heard questions like "Huh?," and "It's supposed to be WHAT?," as well as the occasional, "Wow, I see it!" One question you're sure to hear is, "How do they do it?"

So how do they do it? How do they create the illusion of true depth when we know it's just an ordinary piece of paper with some clever printing on it?

As amazing and seemingly intricate as stereograms are, the concepts behind how a stereogram "works" are actually quite simple. In this chapter, I take you through some of the basic concepts that go into the creation of stereograms. In fact, once you've read through this chapter, you will understand the basic concepts that are used in creating virtually all modern stereogram images.

Let's start with the fundamental purpose of a stereogram. In short, the whole reason for creating and viewing a stereogram is to fool your brain into thinking that it sees depth.

Stereograms are flat, two-dimensional works of art. When you view one properly, however, you'll swear that you are looking at a three-dimensional image!

With the accessibility of high-powered computers and high-resolution printing techniques, commercially available random dot stereoscopic images (stereograms) have evolved into complex, high-quality images. Regardless of their complexity, however, all of these images are created using the same fundamental concepts.

In this chapter, I address some of these fundamental concepts and attempt to explain just how the illusion is capable of fooling your brain into perceiving a third dimension of depth from within a two-dimensional image. I don't attempt to be scientific about this; rather, I present these concepts using a common sense approach. Let's start by taking a layman's look at how we perceive depth.

The Fundamentals

Probably the most important requirement for being able to view three-dimensional depth is to possess good vision in both of your eyes. Have you ever wondered why you have two eyes instead of just one that is located smack in the center of your forehead? Aside from making you much more attractive than a cyclops, having two eyes offers your brain two slightly different views of the world. When you "look" at something, imagine a "ray" coming out of each eye. The two rays intersect, or touch, at the thing that you're looking at. For the sake of discussion, I'll refer to this point where the rays touch as the *point of focus*. Also, I'll call these imaginary lines *line of sight rays*.

NOTE: You've probably noticed that you normally don't see many horses standing around looking at stereograms. Aside from the fact that no one has bothered to create images that the typical horse would find interesting, a horse's eyes are placed too far back on its head to allow it to see easily in three dimensions! The same goes for rabbits, some birds, and most fish.

Two fundamental concepts are at work here, and both involve some information that your eyes provide to your brain. These concepts are used to create stereogram images:

- As an object that you're looking at moves closer or farther from you, your brain will try to keep both of your eyes looking at the object. As the object moves, your eyes move. As the object gets closer, the imaginary line of sight rays coming from your two eyes will be "bent in" more. As the object moves away, the rays will be "bent out." Your brain is very aware of the position of your eyes and this information helps it to judge the distance between you and the object you're looking at.
- Because your two eyes are located at different points on your head, they don't always necessarily see the same thing. This is especially true if you are looking directly at some sort of ledge, where one of your eyes can see the side of the ledge, but the other can't. The reason for this is that the side of the ledge is obscured from one of your eyes by the top of the ledge.

Judging Depth

The first of these concepts involves your brain's ability to judge the distance between you and the thing you're looking at just by the position of your eyes as an object moves closer to you or away from you (and you keep it in focus). The object is forcing your point of focus to some distance away from you.

Now if it were somehow possible to force this point of focus without actually having an object there to look at, and at the same time force your two eyes to see the same thing, it should be possible to fool your brain into judging that there really is something at that point, even if there really isn't.

Stereogram images do exactly this! The repeating patterns that make up a stereogram have been generated in such a way as to force your point of focus to various distances away from you, and at the same time allow your two eyes to pretty much see the same thing. The end result of this is that your brain is fooled into believing that it sees depth where there really isn't any.

How is this done? Actually, it's quite easy. Look at the drawing in figure 2.1. This simple drawing shows an observer looking at an object. I've drawn imaginary line of sight rays between the eyes and the object.

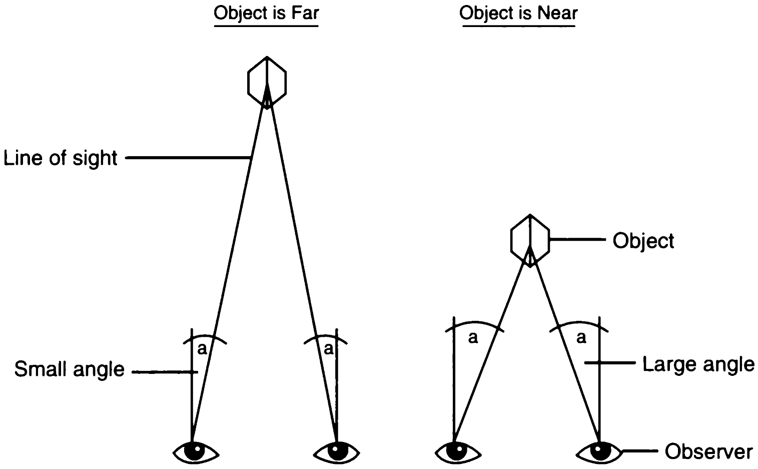


Figure 2.1

Looking at an object.

Look at the angle that is labeled *a* and imagine the object moving away from the observer. As the object moves farther away, angle *a* will get smaller. If the object is moved *very* far away, this angle will effectively go to zero. Similarly, as the object moves closer to the observer, the angle will get larger. If the object moves *very* close, the observer is asking for a headache!

The actual value of the angle isn't important to us in this discussion, but it is certainly important to the observer's brain. By knowing how much these line of sight rays are bent in, the observer is able to make a pretty good judgment as to how far away an object is located. He or she is certainly able to tell if one object is closer than another.

Divergent Viewing

Instead of just a single object for you to look at, what if I had printed a whole row of identical objects on a piece of paper, placed it in front of you and told you to look at the object in the center of the row? (See fig. 2.2.)

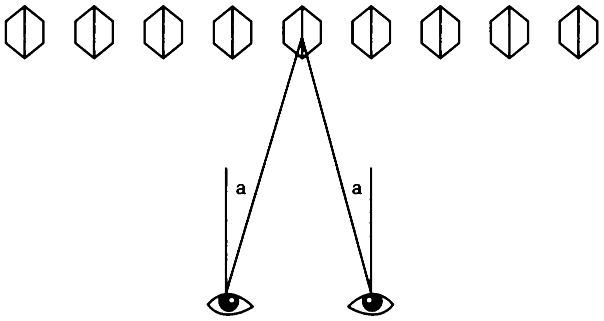


Figure 2.2

Looking at an object in a row.

Now, while you were busy staring at this really interesting object in the center of a bunch of other identical objects, your brain, in noting the positions of your eyes, would be able to judge the distance between you and the paper.

While you're staring at the center object, imagine that someone suddenly put a second piece of paper between you and the first piece of paper, as shown in figure 2.3. Consider what would happen if, on that second piece of paper, there was also a row of objects, spaced as shown in the figure.

Actually, nothing would happen. That's the whole idea! Notice how the line of sight rays from your two eyes each "hit" an object on the second piece of paper. I can now remove the first piece of paper, because you can't see it anymore, anyway.

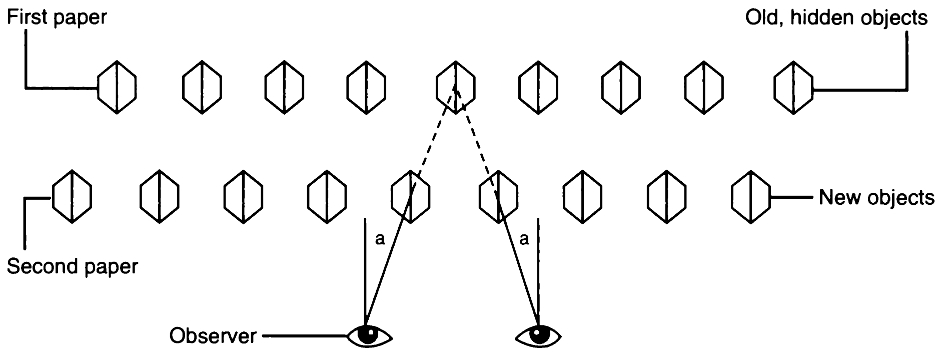
If this were done very quickly, it would be possible to force your point of focus to a point behind the second piece of paper. In reality, this didn't change your point of focus a bit, but think about what has just happened on this second piece of paper, your two eyes are no longer looking at the same object!

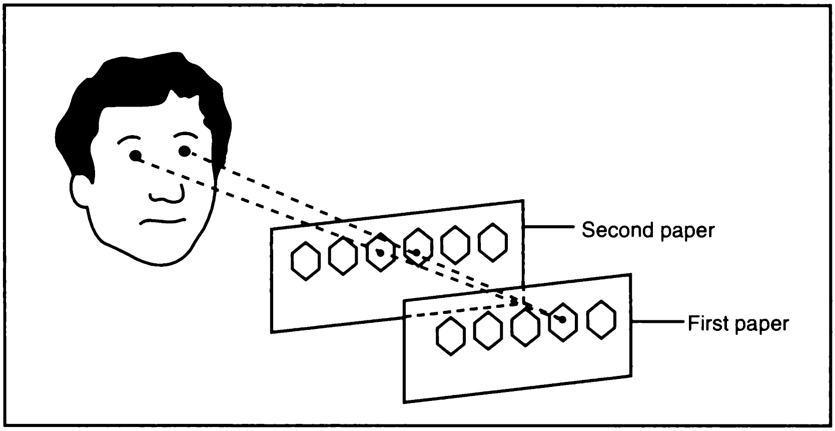
Your point of focus is now behind the second piece of paper (at the same spot it was to begin with). Instead of looking *at* the row of objects, you're really looking *behind* it. Because the objects are all identical, your two eyes each see the same thing, and your brain thinks it is still looking at the center object in the original row.

This sort of viewing is called *divergent viewing* and is essential in being able to properly view a stereogram image. Learning to control your focus so that you are able to look behind an image rather than right at it may take a bit of practice for some people. At the end of this chapter, I've included a collection of hints to help you learn how to do this.

Figure 2.3

Divergent viewing.





Some folks believe that staring at a stereogram causes eyestrain. Actually, this really isn't true. The observer's eyes in figure 2.3 are in reality more relaxed than they would be if he or she were actually focused on the image itself!

Once you've mastered divergent viewing, you're ready to start appreciating stereograms. Technically speaking, the row of objects that is closest to you in figure 2.3 *is* a stereogram! Of course it's not a very interesting stereogram, as I haven't done anything to this repeating pattern of objects to create the illusion of depth.

Creating depth in a repeating pattern of objects like that of figure 2.3 is actually quite easy. It's done by forcing the observer's point of focus to move closer or farther away.

Forcing Focus toward the Viewer

As mentioned earlier, forcing the observer's point of focus toward the observer will create the illusion that an object is closer than other objects in the row.

Stare at figure 2.4 by viewing it divergently (that is, try to look *behind* the row of objects.

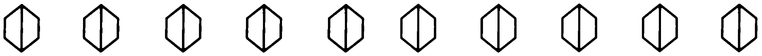


Figure 2.4

Forcing focus toward the observer.

Notice anything? When you view figure 2.4 divergently, you will notice that the center object appears to be closer to you than the remainder of the objects. It actually looks closer to you and it "pops out" of the page!

When you first look at figure 2.4, you may notice that I've placed ten objects in the row. So where's the center object? Think about what happens when this row is viewed divergently. When the line of sight from your left eye passes through the object just left of center, the line of sight from your right eye is passing through the object just right of center. Where do those two lines of sight converge? Right in the center! This is why, when you're viewing the image divergently, you'll see an object directly in the center of the line.

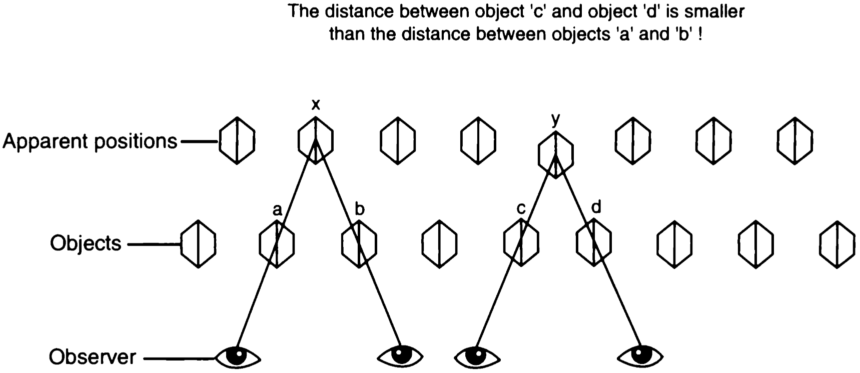
Practice doing this. If you can view figure 2.4 divergently and see that the center object appears to be closer than the remainder of the objects, it will really help you with the remainder of this chapter.

Figure 2.4 uses the same repeating series of objects that I showed in figure 2.3, with one minor exception. If you look closely, you'll notice that the spacing between the middle two objects is closer than the spacing between the rest of them. This figure is a true stereogram. It may not be worth writing home about, but it is certainly an example of a simple image that, when viewed divergently, forces the point of focus of your eyes to multiple distances away from you.

Figure 2.5 is a sketch with the observer's line of sight rays drawn to show what the observer sees.

Figure 2.5

What the observer sees (apparent object position).


The distance between object 'c' and object 'd' is smaller
than the distance between objects 'a' and 'b' !
x
y
Apparent positions
a
b
c
d
Objects
Observer

When the observer's left eye is looking at the object labeled *a,* his right eye is looking at object b. The observer's brain is fooled into thinking it's really looking at an object at point x.

Compare this to when the observer's left eye is looking at point c while his right eye is looking at d. Here, the brain is being fooled into thinking it's seeing an object at point y, which appears closer to him than the object at point x!

The important concept to realize here is that I was able to force your point of focus closer to you as I decreased the distance between two of the objects. This resulted in you seeing one of the objects as appearing to be closer to you than the rest. It is also important to realize that the amount of apparent depth will be directly related to how much the distance between two of the objects is decreased. The closer the distance, the closer the object will appear to be.

Forcing Focus away from the Viewer

Just as your point of focus can be forced closer to you, it can also be forced away from you. Stare at figure 2.6 divergently.

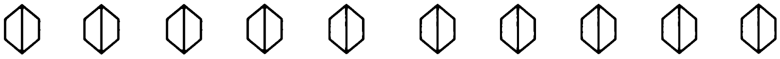


Figure 2.6

Forcing focus away from the observer.

What do you notice this time? What have I done to the pattern? This is the same repeating pattern of objects that I've used previously. The only difference is a minor adjustment in the spacing of the objects, just as in the last example. This time, I increased the distance between the two center objects.

The result is that when the objects are viewed divergently, it appears that the center object is farther away than the rest of them! Just as before, this was done by forcing your point of focus. This time, I forced it away from you.

When the distance between the repeating objects is increased, your point of focus is forced away from you. The result is that your brain can be fooled into seeing the object "pushed in" from the rest of the row.

Figure 2.7 combines these two concepts of "toward you" and "away from you," and shows that they are really the same concept. The point is that by altering the distance between the repeating row of objects, I can selectively move your point of focus closer to you or further away from you, by an amount determined by the spacing of the objects.

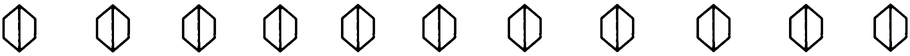


Figure 2.7

Point of focus at multiple distances.

Right/Left Eye Discrepancies

The second major concept involved in the creation of the stereogram illusion is that discrepancies exist between what is seen by your left and right eyes when viewing an image.

When you look normally at some three-dimensional object, your eyes present your brain with two slightly different views of that object.

Your right eye sees a bit more of the right side of the object, and your left eye sees a little more of the left side. This is a *discrepancy,* which simply means that the two views aren't quite identical.

Your brain relies on such discrepancies. These slightly different views of the same object provide your brain with important information in depth perception. Your brain's job is to resolve these discrepancies, and it can do so by realizing that you're looking at a three-dimensional object!

Modern stereogram generation relies heavily on this concept. By providing your eyes with a slightly different view of a scene, the stereogram can force a discrepancy. If this is done properly, the image can force your brain to resolve the discrepancy by believing that it sees a three-dimensional object, when in reality it does not!

This task really isn't as difficult as it may seem. When a stereogram is viewed divergently, the two eyes aren't looking at the same point on the image, as I've shown earlier. Because your two eyes are looking at different points on the image, it really isn't a difficult task to make your two eyes see slightly different things. By making the objects slightly different, a discrepancy is introduced that the brain must resolve; it is fooled into seeing depth that doesn't exist.

Look at figure 2.8 and view it divergently. Notice that the objects seem to have a three-dimensional "feel" about them. Also notice that I've created the objects so that each of them is slightly different from its neighbor. Because you're viewing the line objects divergently, your two eyes see a slightly different view of what your brain thinks is the same object!

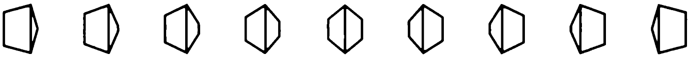


Figure 2.8

Forcing left/right eye discrepancies.

In order to create the tiny changes of depth that are necessary for high-quality, detailed images, modern stereograms are created by using a more complex pattern than just rows of repeating objects.

The Pattern Concept

In the vast majority of all stereograms, the first thing that you may notice is that they are made up of some sort of repeating pattern, and that pattern always repeats in the horizontal (left-right) direction.

The stereogram pattern may be a random set of colored dots, a repeating design of some sort, or even something that originated as a photograph. The point is, whatever is used as the basic pattern, it repeats along the width of the stereogram.

In the simple examples I've presented thus far, a better way to look at them is to see that the pattern actually consists of an object and some blank space, rather than just an object. If you take an object and some blank space, and position it beside another object and some blank space, and keep on repeating it, the result is a bunch of objects, all separated by some blank space!

This concept may sound a bit silly, but it is important. In a stereogram, the pattern repeats, end to end, all along the width of the image. Blank space is considered a part of the pattern just as the repeating object is part of the pattern. We will see later that for high-quality, detailed images, you need to minimize the use of blank space when creating a pattern!

Figure 2.9 is an example of a repeating pattern. Rather than use blank space and an object as the pattern, I've simply used 10 characters; the numbers 0 through 9. I could have used more or fewer characters, and I could have used virtually anything as the actual elements of the pattern. The important thing is that there is a fixed set of elements in the pattern, and all the elements are of the same width.

```
0123456789012345678901234567890123456789012345678901234567890
```

Figure 2.9

A basic pattern.

While this repeating pattern is a bit more sophisticated than simple objects, some readers will immediately notice that it's not much of a stereogram. Because the original pattern is repeated identically across the row, it's not all that interesting, just as a row of equally spaced objects is not all that interesting.

Compare figure 2.9 with figure 2.10. In figure 2.10, I've used the same repeating 0 through 9 pattern as in figure 2.9, but made one minor adjustment—I deleted a zero from the center.

Figure 2.10

The same pattern with a minor change.

```
01234567890123456789012345678912345678901234567890123456789
```

Stare at figure 2.10 divergently. What do you see? When figure 2.10 is viewed properly, nine numbers in the center of the line will seem to be raised above the remainder of the line. These raised numbers are 1 through 9. When you first think about it, this doesn't seem quite right! After all, all I did was delete a single zero from the center of the line. Earlier, when I presented the simple stereograms using repeating objects, moving the two middle objects closer together resulted in seeing just one of the objects closer than the rest. In figure 2.10, I deleted a single character from the pattern, yet the result was that 9 characters now appear closer than the others. What's going on?

Pattern Discrepancies and Apparent Depth

One easy way of explaining figure 2.10 is to say that the deletion of an element of the pattern has caused a discrepancy between what your left and right eyes see.

Because the pattern has been altered from its original 0 through 9 sequence, and because you're two eyes aren't looking at the same spot on the page when you view it in a divergent manner, I've created a discrepancy. Near the center of the line, there are places where your left and right eyes do not see quite the same thing. Your brain resolves this discrepancy by believing that there is a "step," or "edge" in the line.

Figure 2.11 is a sketch of what you see when you view figure 2.10 divergently. The observer in figure 2.11 is fooled into believing that numbers 1 through 9 near the center of the line are closer than the remainder of the line. I've sketched in the line of sight rays to show how the observer is fooled.

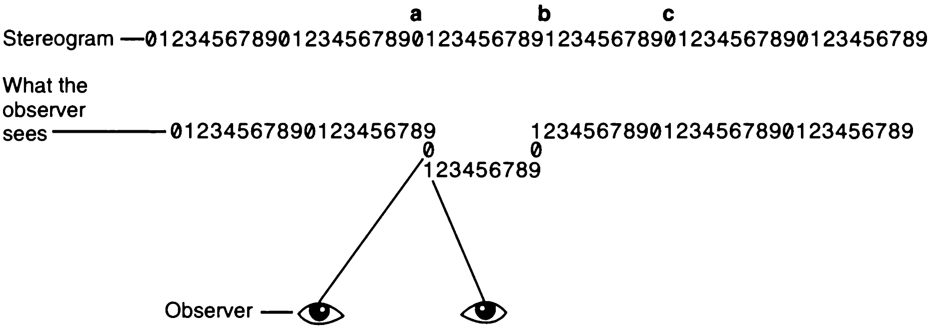

a
b
c
Stereogram —01234567890123456789012345678912345678901234567890123456789
What the observer sees —01234567890123456789
12345678901234567890123456789
0
0
123456789
Observer

Figure 2.11

Simple stereogram with a raised region.

Consider when the observer is staring at the line so that his left eye line of sight is passing through the point labeled as *a*. At the same time, the right eye's line of sight is passing through point b. While the left eye sees the numbers 7890123, the right eye sees the numbers 789123.

Put simply, the observer's left eye sees a zero that the right eye doesn't see (because I've deleted it from the pattern). The easiest way for the brain to resolve this obvious discrepancy is to believe that there is a ledge or step, such that the zero that can be seen by the left eye is obscured from view by the right eye because of the ledge.

Now, consider when the observer glances just a bit to the right, so that the left eye line of sight is near point b while the right eye line of sight is near point c. This time, the right eye sees the numbers 7890123 while the left eye sees only 789123. This is a second discrepancy!

This time, the easiest way for the brain to resolve the discrepancy is to imagine another ledge. This ledge allows the right eye to see a zero, but obscures that zero from the left eye.

Furthermore, consider the first way in which I created the illusion of depth earlier in this chapter. By shortening the distance between two objects, I could force your point of focus closer to you in some parts of the line, fooling your brain into thinking one of the objects is closer than the rest.

Consider the number 1 near the point labeled *a,* and it's distance to the 1 near the point labeled *b.* Because I've deleted a zero, the two 1s are only separated by nine character widths, instead of ten. The same holds true for the 1 at point b and the 1 at point c. These, too, are only nine character widths apart. When you look in this vicinity of the line, your brain is fooled into thinking that the 1 is closer to you than any of the other 1s on the line. The same holds true for the two, three, and so on. In total, nine characters (1 through 9) are separated by a distance of nine characters just by deleting a single zero.

The end result of this is that by deleting one element from the repeating pattern, I was able to cause a discrepancy to occur at two different places. The discrepancy was such that it could be resolved by the brain if there were two ledges obscuring the zero. Furthermore, your point of focus has been forced closer to you when looking at nine of the characters. Combined, these two concepts make it a trivial matter for your brain to believe that it is looking at a raised region, or level of depth. This explanation would resolve both discrepancies, and the closer point of focus would tie all this together. It really looks like the `123456789` is closer to you than the remainder of the line!

This overall concept of starting with an initial pattern of some sort, and modifying the pattern by deleting an element or elements from it at selected points is the basic building block for all stereograms. Because our brains are anxious to resolve little discrepancies in what the two eyes see, it becomes an easy matter to fool an observer into thinking that depth exists where there is none.

In figure 2.11, I've created the simplest region of depth that can be generated. By now, some readers may be asking themselves, "I now understand how to create a single region of depth that is nine pattern elements wide, but how do I create a region that is more or less than nine elements wide?" In the next chapter, you learn some of the more detailed aspects that go into creating a region of an arbitrary width and how that region can be placed anywhere on the line. Also, you will discover how multiple regions are created on a line, as well as how regions are created on top of other regions, resulting in even more depth.

Using the Basic Concepts

Stereogram images are typically a bit more sophisticated than the simple examples that have been provided thus far. The most obvious difference is in the size of the individual elements of the pattern. Instead of starting with a pattern consisting of ten characters, what if we had started with a pattern consisting of 100 tiny dots or even 1000 tiny dots? The exact same concepts would apply, whether the pattern consists of really huge characters, or very tiny dots.

Also, stereograms tend to be a bit more interesting when they consist of more than just a single line!

Line Independence

In all the figures presented so far, the stereograms have been very simple, and were created to illustrate one point. They are all one-dimensional stereograms—they have width, but no height. A normal stereogram that you're used to seeing has both width and height.

NOTE: Of course, the line of characters does have some amount of height. If I took a line of characters and stretched them so that all the characters were a foot tall, they'd have a lot of height. The point is that they would still be examples of one-dimensional stereograms.

In order to create stereograms that are more than a single line high, you may wonder if there are any other things that must be considered.

The good news is that there isn't! Because you've only got two eyes, and because they're oriented horizontally on your head, creating depth in a stereogram is a matter of repeating a pattern only in the left-right dimension. What this means, is that if you really want to get into technicalities, a stereogram is really just a one-line-high image!

When you create a stereogram that has many lines, what you're actually doing is creating many one-line-high stereograms, and then arranging the individual stereograms vertically. In other words, each line of a normal multiline stereogram is independent of all the other lines! A stereogram is literally created a line at a time, and any modifications made to a pattern on one line have no effect on any other line. Each line stands on its own, including the pattern used.

The only important thing is that on each line of a stereogram, the initial pattern always starts off with the same number of elements. The actual elements may be different, but the starting pattern widths are all the same. Figure 2.12 is an example of a multiple-line stereogram. If you look closely at it, you may notice that there are only two different types of lines in the image. I've created the image by stacking the lines of figures 2.9 and 2.10 on top of each other to create a simple stereogram containing a raised square. The center lines of the image are just copies of figure 2.10, while the lines at the top and bottom are copies of figure 2.9.

Figure 2.12

A multiple-line stereogram.

```
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567891234567890123456789012345678901
0123456789012345678901234567891234567890123456789012345678901
0123456789012345678901234567891234567890123456789012345678901
0123456789012345678901234567891234567890123456789012345678901
0123456789012345678901234567891234567890123456789012345678901
0123456789012345678901234567891234567890123456789012345678901
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678901234567890
```

Later in this book, I address some interesting effects of purposely creating stereograms where one line affects some of its neighboring lines, even though it's really not necessary to do so.

It is possible to create stereograms where the final appearance of each line of the image is slightly affected by the lines directly above and below it. This technique is called *filtering,* and is discussed further in chapter 6. Basically, filtering is used to soften the appearance of the stereogram after it has been generated. In the actual creation of the stereogram, each line is processed independent of the other lines.

Basic Pattern Substitution

Figure 2.12 is a true stereogram. Even though the pattern that was used to create the image is really not all that interesting, and the individual elements of the pattern (the numbers 0 through 9) are pretty big as stereograms go, it is indeed a true stereogram.

The trick, now, is to take this simple stereogram and to add a bit of randomness. If the individual elements of the stereogram were dots instead of characters, and the patterns that I used on each line were random dots instead of the sequential numbers, the result would be a random dot stereogram; hence, the name.

The easiest way to create a stereogram that's got a bit of randomness to it is to begin by creating an image similar to figure 2.12, using a sequence of numbers. After each line is processed so that the proper elements are deleted from or inserted to the pattern at the proper locations, each of the numbers in the line can be replaced with a random set of dots, characters, symbols, or just about anything you can think of.

This is called *pattern substitution,* and it refers to the process of replacing each number in the stereogram with something besides a number. This is the final step performed when creating a stereogram. Actual substitutions typically change from line to line.

For example, let's look at the first line of the stereogram. It just so happens that in this first line, no modifications were made to the original 0 through 9 pattern. At this stage, this doesn't matter. The important point is that every line contains ten different numbers, 0 through 9.

Suppose I decided to substitute a random set of ten letters of the alphabet for the ten numbers that were used to create the first line. I pick ten letters at random and list them. I also assign a number to each of the ten random letters, using the numbers 0 through 9.

```
0123456789
KSOWMUDPIZ
```

Now, using this table, I create a line of letters, substituting one of the ten letters for its matching number. All the zeroes are replaced with Ks. All the 1s are replaced with Ss, and so on. Figure 2.13 shows the results of this substitution.

For the second line, I start all over and do the same thing. I start by choosing ten different random letters and create a table:

```
0123456789
UELVJDIFPS
```

Figure 2.14 shows the result of substitution.

Figure 2.13

Pattern substitution.

```
KSOWMUDPIZKSOWMUDPIZKSOWMUDPIZKSOWMUDPIZKSOWMUDPIZKSOWMUDPIZ
```

Figure 2.14

Using a different substitution pattern.

```
UELVJDIFPSUELVJDIFPSUELVJDIFPSUELVJDIFPSUELVJDIFPSUELVJDIFPS
```

It just so happens that the first and second line of figure 2.12 are identical, but that really doesn't matter. For each line, I pick a new set of random letters. I continue to do this for all the lines, and pick a new set of letters for each line (see fig. 2.15).

Figure 2.15

Pattern substitution for each line.

```
KSOWMUDPIZKSOWMUDPIZKSOWMUDPIZKSOWMUDPIZKSOWMUDPIZKSOWMUDPIZ
UELVJDIFPSUELVJDIFPSUELVJDIFPSUELVJDIFPSUELVJDIFPSUELVJDIFPS
MFIEPGIDZOMFIEPGIDZOMFIEPGIDZOMFIEPGIDZOMFIEPGIDZOMFIEPGIDZO
OQJDLVPCUEOQJDLVPCUEOQJDLVPCUEOQJDLVPCUEOQJDLVPCUEOQJDLVPCUE
JORCUARTFMJORCUARTFMJORCUARTFMJORCUARTFMJORCUARTFMJORCUARTFM
GUFVJGOPZNGUFVJGOPZNGUFVJGOPZNUFVJGOPZNGUFVJGOPZNGUFVJGOPZNG
FPODUXYERGFPODUXYERGFPODUXYERGPODUXYERGFPODUXYERGFPODUXYERGF
QPROLGKFJNQPROLGKFJNQPROLGKFJNPROLGKFJNQPROLGKFJNQPROLGKFJNQ
AMGBCUIGPLAMGBCUIGPLAMGBCUIGPLMGBCUIGPLAMGBCUIGPLAMGBCUIGPLA
XODLQPFMNRXODLQPFMNRXODLQPFMNRODLQPFMNRXODLQPFMNRXODLQPFMNRX
BEPRIFMTLKBEPRIFMTLKBEPRIFMTLKEPRIFMTLKBEPRIFMTLKBEPRIFMTLKB
WMADFTPXMNWMADFTPXMNWMADFTPXMNWMADFTPXMNWMADFTPXMNWMADFTPXMN
SITRLKHOPWSITRLKHOPWSITRLKHOPWSITRLKHOPWSITRLKHOPWSITRLKHOPW
RPOTLKHJWSRPOTLKHJWSRPOTLKHJWSRPOTLKHJWSRPOTLKHJWSRPOTLKHJWS
UARTPLMCVIUARTPLMCVIUARTPLMCVIUARTPLMCVIUARTPLMCVIUARTPLMCVI
PALSKJHDGYPALSKJHDGYPALSKJHDGYPALSKJHDGYPALSKJHDGYPALSKJHDGY
```

Try viewing figure 2.15 divergently. You can still see the same raised square! When a computer generates a stereogram, it does so in exactly the same manner. After creating a line that consists of numbers, it performs a pattern substitution and replaces the numbers with something else. This "something else" can be random black-and-white dots, random dots, dots that it obtains from another image, or even a set of random letters as in figure 2.15. Your POPOUT-LITE software creates random dot stereograms using these same concepts. The final appearance of the stereogram depends greatly on how you've directed the software to go about choosing either black-and-white or colored dots for a substitution pattern. In the next chapter, we discuss the concept of pattern substitution in more detail.

Learning to View Stereograms

If you can control your focus in order to properly view stereograms, you may want to skip over this section. I've included a number of hints and tips that are used by various people to help them view stereograms.

If you have trouble viewing a stereogram, read through the following list. Hopefully, one of these tips will work for you.

- Try an old trick that's been around for years. Put the tips of both of your index fingers at the base of your nose, just between your two eyes. Keep the tips of your two fingers touching, and in a straight line with each other. Now move your fingers away from your head. You should see a *phantom finger* that has a fingernail on both ends! Try moving your fingers away from your head, while keeping your eyes focused so that the phantom finger doesn't shrink away. Experiment to see how far away from your head you can move your fingers and still see this phantom finger.
- Start by placing a stereogram about a foot from your eyes. Stare at the image, and try to relax your focus. Imagine that you are looking right through the image. If the patterns start to move, let them. Don't try to adjust your focus; just let the patterns move.
- Place a stereogram behind a piece of clear glass or plastic. Stare at the stereogram, but concentrate on looking at your reflection rather than at the stereogram. Your reflection is located at a point behind the image; this trick is used to force your point of focus behind the image.
- Hold a stereogram about a foot from your eyes, but don't look at it. Instead, look at a point on a wall behind the stereogram. Quickly, now, glance at the stereogram.
- On a flat table, place two pennies, face up, next to each other so that they touch. Make sure that Honest Abe faces directly right on both pennies. Now stare at the two pennies and relax your eyes. When you can see three pennies instead of just two, you're looking at them in a divergent manner so that one eye is looking at each of the pennies. Once you've got this mastered, try

increasing the space between the pennies. Experiment to see how wide you can make the spacing.

Summary

In this chapter, I've tried to present the most basic concepts that are used in the creation of all modern stereogram images. If you've understood these concepts, you're well on your way to a full understanding of what goes into the creation of a stereogram. I've made a list of these major points that I want to make sure you understand from this chapter:

- Possessing normal vision in both eyes allows you to view the world in three dimensions. It gives you the ability to perceive the concept of depth.

- Your brain has the ability to estimate how far away you are from an object that you're looking at. Your brain determines this from the position of your eyes.

- It is possible to look at a horizontal row of identical objects such that your two eyes are not looking at the same object, but rather at adjacent objects. This is referred to as "divergent viewing."

- The natural tendency of your brain is to keep your eyes looking at what it thinks is the same object. This can be "used" to fool the brain into thinking there is depth, where in fact none exists. This is easily done by altering the spacing between some of the objects.

- Shortening the spacing between objects will pull your point of focus toward you, making the viewed object appear to be closer. Lengthening the spacing will push the point of focus away from you, making the viewed object appear to be further away.

- The brain can also be fooled into believing there is depth where none exists by forcing the two eyes to see slightly different things.

- With a continuous, repeating pattern of some kind, deleting an element has the effect of causing a discrepancy between what the left and right eye see. The result will be that a section of the pattern will appear to be closer to you than the remainder of the pattern. This is the basic concept behind building a stereogram.

- When this repeating pattern is substituted with something more random, the result is a true, one-dimensional stereogram.
- Each line of a stereogram is independent of all the others. Creating a two-dimensional stereogram is simply a matter of vertically stacking a series of one-dimensional stereograms.
- There are a number of different techniques for learning to view stereograms. The key is being able to relax your eyes so that you view the image divergently.

In the next chapter, I present a more in-depth look at the specifics of precisely controlling the position, size, and number of levels by modifying the starting pattern. I also explain how to create and use various types of patterns for the last phase of image generation—pattern substitution.

3

Fundamentals of Creating a Stereogram

The previous chapter introduced some of the basic concepts of depth perception, and described how a repeating pattern of elements can be altered in such a way as to fool the brain into seeing depth that really isn't there. This chapter takes a more in-depth look at this concept of pattern modification and examines the specifics of pattern distortion necessary to create arbitrarily sized and positioned regions of depth. Furthermore, this chapter covers various types of pattern substitutions that you can make to create very different types of images.

Keep in mind that the concepts introduced here aren't just some sort of abstract theory. For most of the concepts that are discussed, your POPOUT-LITE software creates random dot stereograms using these exact same techniques.

Effects of Single Element Distortion

In the previous chapter, I introduced you to a simple stereogram created by deleting a single element from one instance of a repeating pattern. I used a pattern consisting of the numbers 0 through 9, and repeated it across the line. Then, I deleted a single zero from the center of the pattern, as shown in figure 3.1.

Figure 3.1

Deleting one element from a repeating pattern.

```
01234567890123456789012345678912345678901234567890123456789
```

The result is that by removing one element from one instance of the pattern, two discrepancies are introduced when the line is viewed divergently. This fools your brain into thinking there are two "ledges." Furthermore, this creates a region in the line where there are nine pattern elements (the numbers 1 through 9) that are separated by a distance of nine character widths. Everywhere else, all the elements of the pattern (0 through 9) are separated by a distance of ten character widths. The combination of these two concepts results in tricking the brain into seeing a region nine character widths wide that is raised above the remainder of the line.

Some obvious questions may arise when looking at this image. How, for example, does one create a region of width that is not nine pattern elements wide? Or, how are multiple regions created on the line? We have seen how a single modification of the pattern results in two edges and creates the illusion of a raised region. Does this mean that it's not possible to create a raised region with only a single edge? An example of this is a line where the entire left half is at one level of depth, and the entire right half is at another apparent level.

Look again at figure 3.1. Until now we've looked at it as a number of instances of a repeating pattern, where each of the instances of the pattern was ten characters wide. In the center of the image, I *deleted* one element of the pattern, resulting in a raised area that was nine elements wide. It turns out that it's no coincidence that the width of the raised region is one less than the width of the original ten-character pattern. If I had started with a nine-element pattern consisting of the numbers 0 through 8, and deleted one of the zeroes at the center of the image, the result is a raised region that is eight characters wide.

Look at figure 3.1 in a slightly different manner. Think of how this line would be created if you were to type it out on a typewriter, starting at the left and continuing to the right.

When you start typing, you've got the starting pattern in mind. In this case, the starting pattern is the numbers 0 through 9. Once you've typed all ten numbers in the pattern, you start over, and keep on doing this until you reach a point near the middle of the line.

Figure 3.2 shows this. At the point labeled *a,* I run out of elements in the original 0 through 9 pattern, so I start over. At point b, I again run out of numbers, so I again start at the beginning of the pattern.

```
          a         b         c        d         e
01234567890123456789012345678912345678901234567890123456789
0123456789
          0123456789
                    0123456789                                Current
                              123456789                       Pattern
                                       0123456789
                                                 0123456789
```

Figure 3.2

Creating the pattern sequentially.

At the point labeled as *c,* where it was again time to start over at the beginning of the pattern, I did so, but I also decided to delete the first element of the pattern. At this point in the left to right processing of this line, the pattern has been changed. It is no longer 0 through 9, as I've deleted the zero from the pattern. From this point on, I want to consider the pattern as consisting of the numbers 1 through 9.

At point d, I again run out of numbers from my new 1 through 9 pattern, and it's time to start over. Remember, my pattern is now the numbers 1 through 9, not 0 through 9. This means that at point d, I *insert* a new element into the pattern! Starting at point d, I insert a zero into the current pattern, so that instead of 1 through 9, it is now 0 through 9. At point e, I again run out of numbers in my new pattern, and need to start over. When I start over, I start with the beginning of my pattern, which is now 0 through 9.

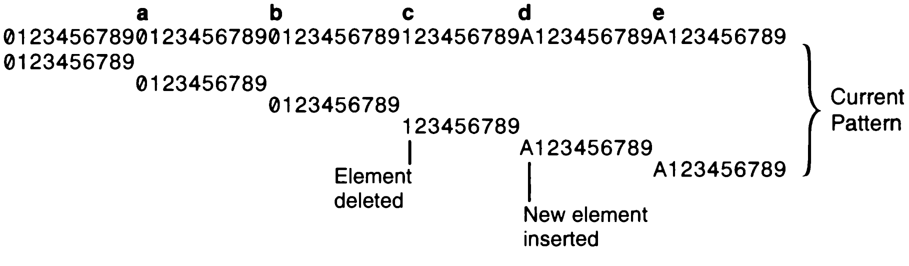
Look familiar? The pattern in figure 3.2 is identical to the pattern in figure 3.1! The point is that if I create this line a character at a time, left-to-right operation, and think of a current or dynamic pattern that may change as I progress across the line, I really haven't made one modification. Actually, I've made two modifications to the pattern! At point c, I modified the pattern by deleting an element from it, shortening the current pattern from ten to nine elements by deleting the zero. At point d, I made a second modification to the current pattern. This time, I modified the pattern by inserting a new element, lengthening the pattern by one element.

This new way of looking at how the line was created is very important in creating a stereogram. When I look at it this way, the explanation of the nine-element raised region is a little different. Moving left to right, a shortening of the current pattern at point c caused the pattern elements that follow to the right to appear to be "popped out" relative to the elements to the left. Nine elements later, a lengthening of the pattern caused the pattern elements that follow to be "pushed in," back to the original level.

Figure 3.3 shows a more general way of looking at this same image. I start out at the left with the same 0 through 9 pattern. At point c, I delete a zero from the pattern, shortening the pattern to the numbers 1 through 9. At point d, I lengthen the pattern by inserting a *new* element into the pattern, namely, the character *A*. Starting at point d, the pattern is now A123456789. At point e, where I run out of pattern and have to start over, I use the current A through 9 pattern. Notice that once I delete the zero at point c, I consider it long gone, and never use it again.

Figure 3.3

Inserting new elements.



```
          a         b         c        d         e
012345678901234567890123456789123456789A123456789A123456789
0123456789                                                    }
          0123456789                                          }
                    0123456789                                } Current
                              123456789                       } Pattern
                              |        A123456789             }
                         Element|                 A123456789  }
                         deleted       |
                                       New element
                                       inserted
```

You may notice that the resulting stereogram is identical to the stereogram of figure 3.2. The difference is that I've used a new philosophy when it comes to deleting and inserting elements from or into the pattern. Once an element is deleted from the pattern, it's gone forever. When the time comes to insert an element into the pattern, I use a *brand new* element rather than use old, previously deleted elements. Practice this philosophy when you create random dot stereograms.

Notice that you can look at figure 3.3 in yet another way. I created this line by starting at the left, and began with the pattern `0123456789`. Because we normally read from left to right, it's easy to look at it that way. An equally valid way of looking at the line is to think of it as starting at the *right,* with a pattern of `987654321A`. If I start from the right and work toward the left, at point d I *delete* the `A` from the pattern, and the pattern becomes `987654321`. This causes the elements that follow to the left to pop out. At point c, I *insert* a new element into the pattern, namely, the character `0`. This causes the elements that follow to the left to be pushed in. From here and until the left of the line, the pattern is now defined as `9876543210`.

This symmetry of being able to process a line either from right to left or from left to right is important in the generation of certain stereograms. For now, and for the sake of simplicity and consistency, I assume that we do all processing from left to right.

Controlling Edges and Widths

In the previous examples, I've shown you that a better way to look at how a stereogram is created is to look at each line as being created one pattern element at a time, progressing from one side of the line to the other. I begin with an initial starting pattern. Then, by deleting an element, I am able to begin a lifted region. A few elements later, I terminate the region by inserting a new element into the current pattern. This causes the remainder of the line to be pushed back to its original apparent level.

This new way of looking at how the line is generated allows some greater flexibility as to the widths of the regions that are generated. Figure 3.4 shows an example where I again start out with the same 0 through 9 pattern. At point c, I again shorten the pattern by deleting the zero and make the current pattern `123456789`. In figure 3.4, however, I continue this new 1 through 9 pattern throughout the remainder of the line. As you can see, I never insert new elements into the pattern.

Figure 3.4

One deletion, no insertions.

```
          a         b         c
012345678901234567890123456789123456789123456789123456789
0123456789                                                  } Current
          0123456789                                        } Pattern
                    0123456789
                              123456789
                              |        123456789
                              |                 123456789
                              Element deleted
```

View figure 3.4 divergently. What do you see? Shortening the current pattern at point c causes the remaining elements to the right to pop out. After deleting the zero from the pattern and redefining it as 1 through 9, I never change it again. By never inserting more elements, there is nothing to push the remainder of the line back in! The result is that the entire right half of the line appears to be closer to you than the entire left half.

You can also create a single edge of a raised region. One edge is simply created by one pattern modification. Figure 3.1 really has two pattern modifications (one deletion and one insertion); the result is two edges.

Understanding this process is essential in understanding how you can control the position and width of any region or regions within a line. For example, in figure 3.5, I start out with the same 0 through 9 pattern. At point a, I decide that it is time to create a level and I delete an element from the pattern. Never mind that point a is in mid-pattern. It really doesn't matter as long as I handle things carefully. Creating the beginning of a raised region simply means that I need to delete an element of the pattern at this point and move on. It just so happens that there happens to be a five right where I wanted to start a raised region.

Figure 3.5

Redefining the current pattern.

```
                         a        b
01234567890123456789012346789012346789012346789012346789
0123456789                                                  } Current
          0123456789                                        } Pattern
                    0123456789
                         678901234
                         |        678901234
                         |                 678901234
                         Element deleted
```

How is this done? Go ahead and delete the five; you redefine the current pattern as you do so. Up until this point, the pattern has been defined as 0123456789. At point a, I delete one element from the pattern, leaving a current pattern consisting of nine elements. After deleting the five, the next element in line is the six. What if at this point I redefine the pattern to now be the nine numbers, 678901234? Notice that the five is gone, and the new, redefined pattern starts out by picking up with the next character in line, namely the six. It then continues on to nine, starts over with zero, and continues to four. I've got the same sequence that I started with; I've just deleted the five. Also, because I deleted an element from the middle of the pattern, I simply made things easier for myself by dynamically redefining the pattern.

At point b, I just type a four, meaning that I've hit the end of the newly defined pattern, and it's time to start over. So, I start over with a six, which is now the first element in the new pattern. I continue throughout the remainder of the line using this new pattern.

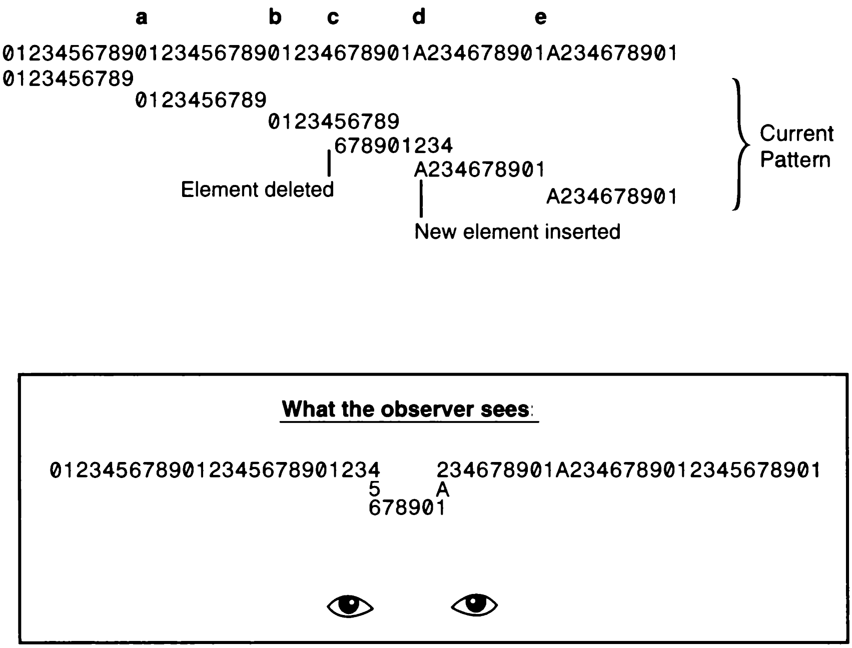
You may have noticed that the right most character in the line is a nine instead of a four. Since the last character in the redefined pattern is a four, shouldn't the last character in the line also be a four? Actually, it doesn't matter. I stopped with the nine simply because I didn't want to make the line any longer. If I were to continue with the current pattern, the raised region would simply continue until the point at which I stop.

You may notice that figure 3.5 is really the same type of image as figure 3.4. In both cases, there is a point in the line where I decide to start a raised region. I do this by deleting an element in the pattern at the spot where I want the raised region to begin. By deleting an element, I redefine the current pattern. In figure 3.4, I delete a zero and redefine the pattern by starting with the next element in line, namely the one. The new pattern is then one less element in length. In figure 3.5, I do exactly the same thing really. The only difference is that I happen to be in mid-pattern when I decide that I want to start a raised region. It really doesn't matter, however, because I simply redefine the pattern by deleting the five, and start the new pattern with the next element in line, namely, the six. In both cases, I repeat the newly defined pattern to the end of the line.

Just as figure 3.5 shows that I can begin a raised region at any arbitrary point in the pattern, figure 3.6 shows that a raised region can also be terminated at any arbitrary point in the current pattern.

Figure 3.6

Terminating a general region.



```
          a         b    c     d         e
012345678901234567890123467890 1A234678901A234678901
0123456789
          0123456789
                    0123456789                          Current
                         678901234                      Pattern
                         |     A234678901
             Element deleted   |         A234678901
                               New element inserted
```

What the observer sees:

```
0123456789012345678901234    234678901A23467890123456789 01
                        5    A
                        678901
```

Figure 3.6 starts out on the left looking just like figure 3.5. At point c, I start a raised region by deleting a five from the original 0 through 9 pattern, and redefine the current pattern as `678901234`. I start to repeat the pattern, but at point d decide that it is time to end the raised region.

I do this by inserting a new element into the pattern, and again redefine the pattern. At point d, I terminate the raised region by inserting a new element `A` into the pattern. Notice that I just typed a one. The next character that I should have typed was a two, but instead, I insert the `A`. What this means is that I again redefine the pattern, this time as `A234678901`.

This new pattern begins with the element that I inserted, namely, the `A`, and continues with the next element waiting in line that should have gone there, namely the two. The new pattern continues until the four at the end, then starts over with the six. Notice that there is no five anywhere. Once it has been deleted back at point c, I consider it to be gone forever.

At point e, I hit the one, the last element of the newly redefined pattern, so I start over with the A, the first element. I continue for the remainder of the line with this new pattern.

Let's examine closely what happened here. When I introduced the first modification into the pattern at point c to start the raised region, the pattern was redefined as 678901234. If you look at figure 3.6 divergently, you can see that the six is the first element in the raised region. Now, after typing only the 6, 7, 8, 9, 0 and 1, I decided to terminate the region, and did so by inserting the A, and redefined the pattern as A23467890.

Between the time that I first modified the pattern by deleting an element, and the second time I modified it by inserting the A, I had typed the six characters listed above. Notice that the raised region in figure 3.6 is six characters wide, and contains these six characters.

The point to all of this is that the width of a raised region in a line of a stereogram, as well as the locations of starting and ending points of the region, is easily controllable. Raised regions are started by deleting an element in the pattern at the desired point where the region is to start. The pattern is then redefined to accommodate this deleted element. Some elements later (that is, the desired width of the region), it is terminated by inserting a new element into the pattern, and again redefining the pattern to accommodate the newly inserted element.

Multiple Regions within a Line

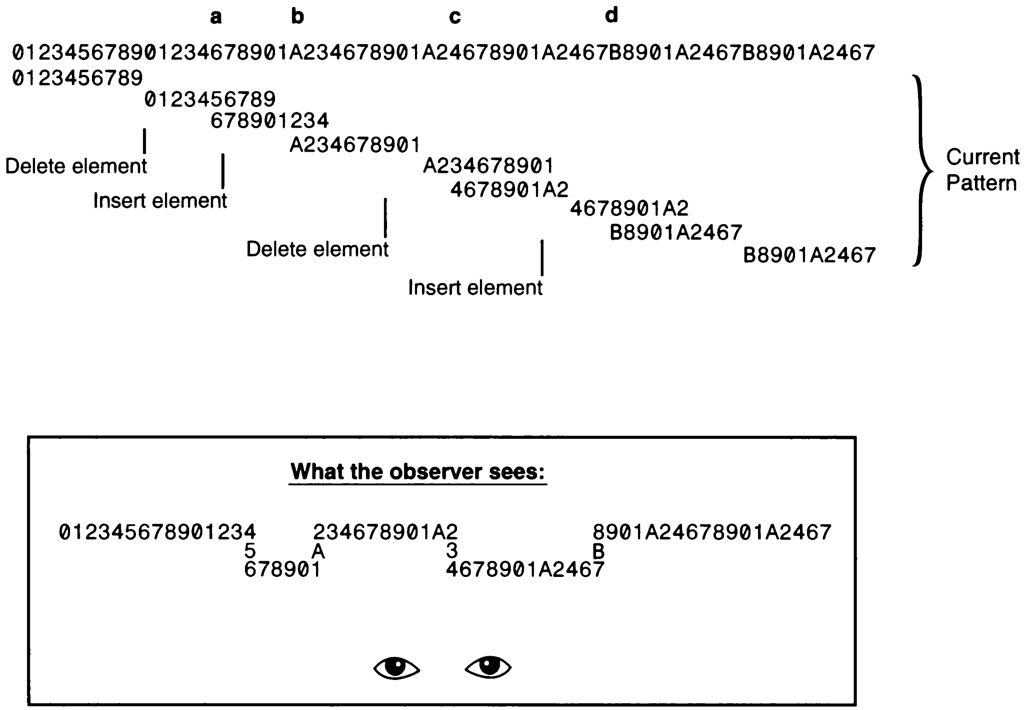
In general, any number of raised regions may be within one line of a stereogram. Figure 3.7 shows a one-line stereogram with two separate raised regions. At first it may look a bit more difficult, but I will show you that in reality it is simply an extension of the technique used to create figure 3.6.

As I've done in the previous examples, figure 3.7 begins at the left with the 10-element pattern of the characters 0 through 9. At point a, I delete a five from the pattern, and redefine the new pattern to be 678901234. Six characters later at point b, I terminate the region by inserting an A into the pattern. At this point, I redefine the pattern as

A234678901. The result is a six-element wide raised region. I continue until point c, where I decide it is time for another region. This time, I delete the three at point c to begin the second region. Using the same concept of redefining the pattern like I did at point a, the new pattern becomes 4678901A2. I want to make this second region a bit wider, so I continue with this newest pattern for 12 characters. At point d, where there should have been an eight, I insert yet another new element. This time, I insert a B. At this point, the pattern again is redefined as B8901A2467. I then continue with this pattern for the remainder of the line.

Figure 3.7

Multiple raised regions.



```
               a     b           c           d
012345678901234678901A234678901A24678901A2467B8901A2467B8901A2467
0123456789                                                          )
          0123456789                                                |
               678901234                                            |
          |           A234678901                                    |  Current
Delete element  |               A234678901                          |  Pattern
     Insert element               4678901A2                         |
                           |               4678901A2                |
                 Delete element               B8901A2467            |
                                 |                      B8901A2467  )
                          Insert element
```

```
                     What the observer sees:

012345678901234    234678901A2          8901A24678901A2467
              5    A         3          B
              678901         4678901A2467
```

It really doesn't matter how wide the regions are, or how many of them there are. As long as we maintain this concept of the current pattern, it's easy. To begin both of the regions was simply a matter of deleting an element in the current pattern at the point where I wanted the region to begin. Terminating the regions was just as easy. At the point that I wanted the raised region to end, I simply introduced a new element into the pattern.

Figure 3.7 shows one region that is six elements wide, and a second that is 12 elements wide. It really doesn't matter if the region width is larger or smaller than the starting pattern width. When it's time to terminate the region, you simply insert a new element, and in doing so redefine the pattern starting from that point.

NOTE: It's important to throw away deleted elements from the pattern. To begin both regions, I deleted an element from the current pattern. Once deleted, that element never again appears in the remainder of the line.

Introducing new elements into the pattern when terminating a raised region is also very important. To terminate both regions, I introduce brand new elements (the A and B characters) rather than reuse old, previously deleted elements.

This concept of not reusing previously deleted elements when inserting new elements into the current pattern is important for generating high quality RDS images. Brand new elements tend to add more of a random feel to the image than do old, deleted ones.

I will show you later that reusing deleted elements can actually be a desirable and necessary way to do element insertions for some types of stereograms. In these cases, refer to figure 3.1, where I did just that. When it came time to end the raised region, I inserted a zero that had previously been deleted.

You can see that creating multiple regions of depth within a line of a stereogram is really no more difficult than creating a single region. The regions are created one at a time, and "on the fly" as the line is processed. It is crucial to redefine and maintain the current pattern as processing progresses across the line. The pattern must be redefined every time a region is started or terminated.

Creating Levels on Top of Levels

In chapter 2, I showed you a simple stereogram containing a single raised square. I explained that because each line of a stereogram is processed independently, a real two-dimensional stereogram is really just a collection of independently processed lines, all of which use the same starting pattern width.

Earlier in this chapter, I showed you that this figure was really just a special case of a more general technique. By processing each line from one side to the other, multiple regions can easily be created on each line. This makes it easy to create more complex images than just a simple square.

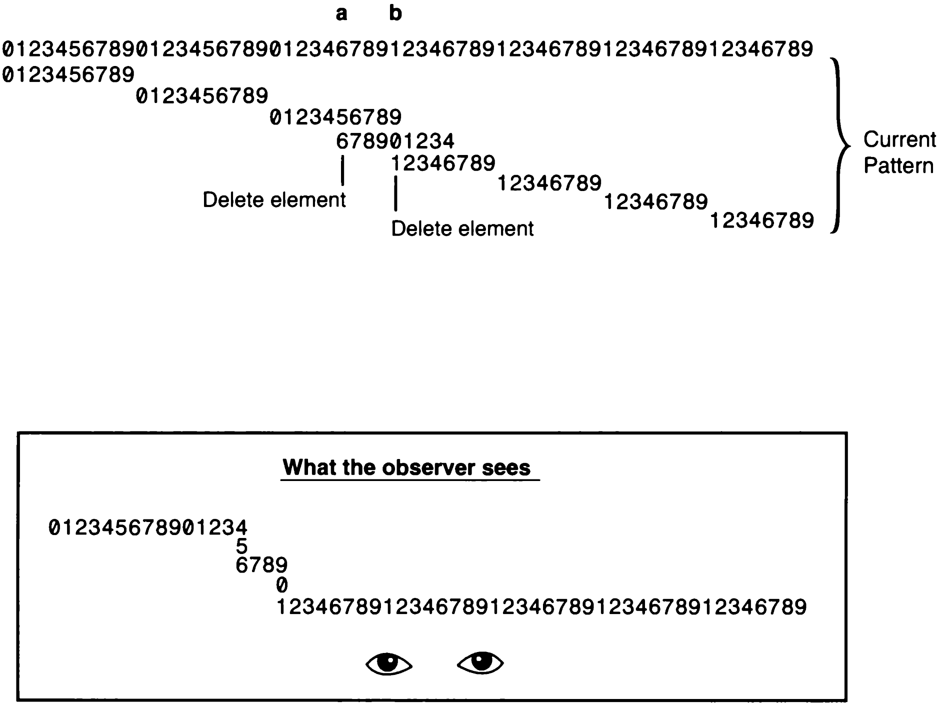
One of the most noticeable things about a typical stereogram is that not only do they contain many regions on a given line, but they also contain many different apparent levels of depth in the image. If you look at a single line of a typical stereogram, you notice that it has more than a single level of depth.

How is this accomplished? Actually, the process isn't difficult at all. At this point, you have all the knowledge necessary to create stereograms with multiple levels of depth.

Let's go back to our one-line stereogram example and again examine what happens when we create a raised region. As I type the characters of the starting pattern in the stereogram, I begin a raised region by deleting an element from the pattern, and redefine the pattern to accommodate the deletion. I showed you earlier that if I continue to repeat this new, shorter pattern, the remainder of line will be raised.

What if, after deleting an element, I proceed by repeating the new pattern, and then decide to delete an element from this new pattern? If I follow all the same rules, it means that I would again have to redefine the pattern to yet another pattern to accommodate this second deletion. Look at figure 3.8 divergently. In figure 3.8, I've made the deletions I just described. What do you see?

Figure 3.8 is our first example of a stereogram containing more than one level of depth. Notice that I start out with the same old ten-element starting pattern of 0 through 9. At point a, I create the left edge of a raised region by deleting a five from the pattern, shortening and redefining it as `678901234`.



```
                         a   b
0123456789012345678901234678912346789123467891234678912346789
0123456789
          0123456789
                    0123456789
                         678901234                              Current
                         |   12346789                           Pattern
                         |   |   12346789
           Delete element    |           12346789
                             Delete element      12346789
```

```
                 What the observer sees

012345678901234
              5
              6789
                 0
                 1234678912346789123467891234678912346789
```

Figure 3.8

Starting multiple raised regions.

I continue with this new nine-element pattern, until I reach point b. At this point, instead of typing the zero, I delete it from the pattern. With the zero deleted, I redefine the pattern as 12346789. Notice that the pattern is now only eight elements wide because I've deleted two elements without inserting anything.

This second element deletion has the same effect as the first one. It creates a discrepancy that the brain has to resolve, and results in you seeing an edge for each of the deletions. Also, with the pattern starting with ten elements, then shortened to nine, and finally eight elements, your point of focus is forced closer and closer to you.

NOTE: Important! For each shortening of the pattern, a raised region is created that is proportional to the number of elements deleted. The more elements that I delete, the closer to you the levels of depth appear to be.

Once I shorten the pattern to eight elements, I continue for the remainder of the line just repeating these same eight elements. Because I never insert anything into the pattern, this second region of depth continues for the entire right side of the image.

Figure 3.9

Levels on top of levels.

```
                         a   b         c    d
0123456789012345678901234678912346789 12A3467B8912A3467B8912A3467
0123456789
          0123456789
                    0123456789
                         678901234                                  Current
                    |        12346789                               Pattern
                    |   |            12346789
         Delete element |              A34678912
              Delete element      |         B8912A3467
                        Insert element |              B8912A3467
                             Insert element
```

```
                  What the observer sees:

012345678901234                          8912A3467B8912A3467
              5                          B
              6789        3467
                 0        A
                 1234678912
```

Compare figure 3.9 with figure 3.8. Do you see any differences? Figure 3.9 starts out identical to figure 3.8. I start with a ten-element pattern, and then shorten it to nine at point a, and finally eight elements at point b. At point b, the current pattern is defined as 12346789. From point b, I continue typing for a few characters, and then decide to insert an element into the current pattern. I do this at point c, where I insert an A, and redefine the pattern to A34678912. This insertion creates a right-hand edge, and causes the level of depth to push away from you, because the pattern width has increased from eight to nine elements. A bit later, at point d, I insert a second element, a B. I redefine the pattern as B8912A3467. I then repeat this 10-element pattern for the remainder of the line.

Just as the insertion of the A at point c causes a push in of the apparent depth, a second element insertion of the B at point d causes a second push in, back to the original level as on the left side of the line. This makes sense, because inserting the B brings the pattern width back up to ten elements, the same width that I started with at the left.

Notice the widths of the various levels of depth. At point a, where I delete the five from the pattern and redefine the pattern as 678901234, I start the first region, or level. I type a 6, 7, 8, and 9 before I do the second deletion at point b. Notice that this first level is four characters wide and contains the characters 6789. At point b, I delete the zero from the pattern, creating the beginning of the second level, and the pattern is redefined as 12346789. Then, I type the 10 characters 1234678912 before doing the insert at point c. The level closest to you is 10 characters wide, and contains these characters. After inserting the A at point c, I type four more characters before doing the second insert at point d. With the pattern redefined as A34678912, I type 3467 before I insert the B. Notice that those four characters appear on the pushed-in level created by the insertion of the A, and before the insertion of the B.

As you can see, creating levels of precise widths is easy to do, once you understand how it's done. This same concept can be extended to producing three, four, or a thousand different levels! It's all done the same way. To begin another level of depth, simply delete an element from the current pattern. To end a level of depth, simply insert an element.

In figure 3.9, I created a total of three levels of apparent depth. This was done by the three different current pattern widths I encountered as I created the line. I went from the bottom to the middle level by deleting an element, and then from the middle to the top level by deleting a second element.

What if I had just gone and deleted both elements at once? Figure 3.10 shows what would happen. Do you notice any differences between figure 3.10 and 3.9?

Figure 3.10

Controlling distance between levels.

```
                         a           b
0123456789012345678901234789012347890AB12347890AB12347890
0123456789
          0123456789
                    0123456789                                Current
                         78901234                             Pattern
                         |       78901234
                         |           AB12347890
           Delete two elements       |         AB12347890
                                     |
                        Insert two elements
```

```
                What the observer sees:

012345678901234          12347890AB12347890
              5          B
              6          A
              789012347890
```

In figure 3.10, I again start with the same 0 through 9 pattern. At point a, however, I decide to create a level; but instead of deleting a single element from the pattern, I delete two elements, namely the five and six. With these two elements gone, I redefine the pattern as `78901234`. In one fell swoop, I went from a 10-element pattern to an eight-element pattern. I continue this shorter pattern until I reach point b, where I decide to end the region. Instead of just inserting one element, I insert two, an `A` and a `B`. At this point, I've redefined the pattern to be `AB12347890`. Notice that I'm back to a ten-element pattern.

As you can see, the result of doing this is that since I did deletions there in only one spot, and I did insertions in one spot, I created only one raised region. But, what a raised region! By deleting two elements at once (then, inserting two later), I created a raised region that is *very* raised. If you compare this to figure 3.9, you can see that the apparent depth of the region is identical to the top level of figure 3.9.

NOTE: Important! The number of pattern elements that are deleted or inserted helps to determine the apparent depth between the levels. The more levels that are deleted or inserted, the more drastic the change in apparent depth will be.

With the examples I'm using here, I've got really big elements in my pattern, and there aren't a whole lot of elements in the pattern. Imagine if, instead of using characters for pattern elements, we use very tiny dots. You can pack a lot of dots in the space it takes me to type ten characters! Every time one of the dots is deleted from the pattern, a level of depth is created, and every time a dot is inserted, the level is terminated.

Later in this book I go into some of the details, but for now I want you to realize simply that the apparent distance *between* the different levels is proportional to the size of the elements that make up the pattern. Because my pattern elements are pretty big, the apparent spacing between the levels in figure 3.9 is also pretty big. If the pattern elements are tiny dots, each deletion or insertion of a pattern element results in a proportionally tiny change in the apparent depth of the level created.

In a true high-resolution stereogram, the dots that make up the pattern are very small, and the different levels appear to be very close together. They are so close, in fact, that in many stereograms it's very hard to distinguish the individual levels at all!

You can think of a stereogram as being made up of stacks of different apparent levels, something like a topographic map. Each of the objects in a stereogram is actually made up of a stack of levels with slightly different shapes and sizes.

Try this experiment. Stare at the color stereograms included in this book and see if you are able to identify the levels that are stacked to create the image. In many images, it's not easy.

When the elements of the pattern are very small, and the corresponding apparent distance between levels is also very small, the human brain tends to integrate the edges of these different levels. This means that it tends to smooth out the edges, so that instead of seeing the edges of the individual levels, it sees a gradual changing in depth, and the edges of the stack really appear to be a surface that extends in three dimensions.

This is the trick behind stereograms! Very small dots can allow for lots of dots in a pattern, which means that there can be lots of insertions and deletions of pattern elements. This means lots of levels. Very small dots also mean that the individual levels appear to be very close together, making them very difficult to see. Instead, your brain tends to smooth out the edges of the individual levels, and you tend to see actual three-dimensional objects rather than just a topographical stack of individual levels.

Creating Multiple Lines

As shown in the previous chapter, each line of a stereogram is processed and generated independently of all other lines in the image. Each line is technically a stereogram all by itself. But as in the examples I've been using, a one-line stereogram just really isn't very interesting.

We can combine some of the concepts that have been presented in this chapter, and combine one-line stereograms to create something a bit more interesting. Figure 3.11 does this. In this example, I create each of the lines by starting with the same ten-element pattern of the characters 0 through 9. In figure 3.11, can you pick out examples of the lines described below?

- Lines that have no changes in depth. I simply repeat the original pattern all the way across without doing any insertions or deletions to the pattern.
- Lines with a single, slightly raised region. To create lines like these I deleted one element of the pattern to begin the region, and then inserted a new element to terminate the region.
- Lines that have a single, significantly raised region. I deleted two elements of the pattern to begin the region, and inserted two new elements to terminate the region.
- Lines that have more than one raised region.
- Lines with regions at different levels.

```
012345678901234567890123456789012345678901234567890123456789
012345678901234567890123456789012345678901234567890123456789
012345678901234567890123456789012345678901234567890123456789
012345678901234567890123456789012345678901234567890123456789
01234567890123456890123456890A123456890A123456890A123456890A
01234567890123456890123456890A123456890A123456890A123456890A
01234567890123456890123456890A12345680A12345680A1B2345680A1B
01234567890123456890123456890A12345680A12345680A1B2345680A1B
012345678901234568901234568012345680123A45680123AB45680123AB
0123456789012345678901234569012345690 12A345690 12AB345690 12AB
```

Figure 3.11

Combining the various concepts.

Pattern Substitution Types

While figure 3.11 is an interesting stereogram because it shows some of the important concepts that have been introduced in this chapter, it is not very interesting in the sense that I've used the original, numerical pattern in displaying the image.

As shown in chapter 2, a much more interesting stereogram can be created by substituting these pattern values with something a bit more interesting. Coming up, I discuss some of the types of pattern substitutions that are typically made when a stereogram is created. I discuss this in more detail than in chapter 2, and explain some of the techniques that are used in creating a pattern for this substitution phase of stereogram generation.

In this chapter, I've shown you how to create regions of depth by inserting and deleting elements from the current pattern when processing a line of a stereogram from one edge to the other.

Also, you may recall that when creating random dot stereograms, you can obtain better results if you do not reuse deleted pattern elements. In the simple stereogram of chapter 2, this really didn't matter much, but for more complex stereograms, deleted elements really shouldn't be reused.

In figure 3.11, the lines that contain no raised regions have a total of ten pattern elements. Other lines, where raised regions have been created, new pattern elements were introduced. For consistency, I've been calling the first element that I insert into the pattern A, and the second one (when there is one) B. On these lines, I need to know the total number of elements that have been used on the line. On a line that has, for example, two raised regions, I will have inserted an A to terminate the first region, and a B to terminate the second. On a line like this, I've used a total of 12 different pattern elements. Ten of the elements were the original 0 through 9, plus the A and B make 12 elements.

Making Random Character Substitutions

The concept of using random characters when doing the pattern substitution was first introduced in chapter 2. This doesn't result in a very high-resolution, high quality stereogram, but it demonstrates the concept nicely.

In figure 3.12, I've done a substitution of the original pattern elements of figure 3.11, in much the same way as I did back in chapter 2. The difference is that with the simple example in chapter 2, there were a total of only ten different pattern elements used in creating any line of the image. With figure 3.12, there are lines that have more than ten different pattern elements. Can you pick out the lines that have more?

With each line of figure 3.12, I start by counting the total number of elements used on the line. I then make a little substitution table, and proceed to substitute each of the original pattern elements in 3.11 for the random set of letters in my table. For each line, I choose a different set of random characters, thereby keeping a real random look to the image.

```
YNMQPCRTUOYNMQPCRTUOYNMQPCRTUOYNMQPCRTUOYNMQPCRTUOYNMQPCRTUO
XEGCCKPAZVXEGCCKPAZVXEGCCKPAZVXEGCCKPAZVXEGCCKPAZVXEGCCKPAZV
MYUHGLIDJPMYUHGLIDJPMYUHGLIDJPMYUHGLIDJPMYUHGLIDJPMYUHGLIDJP
PMBQFMSFOUPMBQFMSFOUPMBQFMSFOUPMBQFMSFOUPMBQFMSFOUPMBQFMSFOU
EKXNGVADEMEKXNGVAEMEKXNGVAEMEZKXNGVAEMEZKXNGVAEMEZKXNGVAEMEZ
SXFQWBMIAWSXFQWBMAWSXFQWBMAWSTXFQWBMAWSTXFQWBMAWSTXFQWBMAWST
EAIFGZVJGREAIFGZVGREAIFGZVGREDAIFGZVGEDAIFGZVGEDAKIFGZVGEDAK
YMYRVOEQPIYMYRVOEPIYMYRVOEPIYDMYRVOEPYDMYRVOEPYDMDYRVOEPYDMD
BWLRTUQNMXBWLRTUQMXBWLRTUQMBWLRTUQMBWLRXTUQMBWLRXHTUQMBWLRXH
ANLRDOQUZFANLRDOQUZFANLRDOQFANLRDOQFANLMRDOQFANLMQRDOQFANLMQ
AHJGKJGMKRAHJGKJGMKRAHJGKJGRAHJGKJGRAHJNGKJGRAHJNAGKJGRAHJNA
LSXVFVVVNMLSXVFVVVNMLSXVFVVMLSXVFVVMLSXMWVFVVMLSXMWVFVVMLSXM
BFVDNDJMMUBFVDNDJMMUBVDNDJMUBVDNDJMUBVDDANDJMUBVDDANDJMUBVDD
GBRJSVLOYCGBRJSVLOYCGRJSVLOYCGRJSUVLOYCGRJSUVLOYCGRJSUVLOYCG
OXENOLKIBSOXENOLKIBSOENOLKIBSOENOULKIBSOENOULKIBSOENOULKIBSO
WSFCGIXJQIWSFCGIXJQIWFCGIXJQIWFCGRIXJQIWFCGRIXJQIWFCGRIXJQIW
VHORWKCBKQVHORWKCBKQVORWKCBKQVORWNKCBKQVORWNKCBKQVORWNKCBKQV
JMATDSCZTBJMATDSCZTBJMATDSCZTBJMATDSCZTBJMATDSCZTBJMATDSCZTB
GGYBTURQCTGGYBTURQCTGGYBTURQCTGGYBTURQCTGGYBTURQCTGGYBTURQCT
MCMYVANNNGMCMYVANNNGMCMYVANNNGMCMYVANNNGMCMYVANNNGMCMYVANNNG
RXFAKDRWZTRXFAKDRWZTRXFAKDRWZTRXFAKDRWZTRXFAKDRWZTRXFAKDRWZT
```

Figure 3.12

Random character pattern substitution.

An interesting note is that this set of substitution characters doesn't have to be random. For example, on a line of the image of figure 3.11 that uses 12 different pattern elements, I could have created a translation table like the following:

```
0123456789AB
HIDDENIMAGES
```

There are a couple of important concepts here. First, the characters that I choose when doing my substitution don't necessarily have to be random. In the little translation table above, they're obviously not. The second concept is that there is nothing that says I can't use some of the characters more than once. Notice that a D is substituted for both two and three, I is substituted for both one and six, and E is substituted for both four and A.

This sort of nonrandom substitution helps add a bit of a theme to the stereogram. Figure 3.13 is also created using figure 3.11, but is somewhat more interesting.

Figure 3.13

Non-random character pattern substitution.

```
HIDDENIMAGHIDDENIMAGHIDDENIMAGHIDDENIMAGHIDDENIMAGHIDDENIMAG
IDDENIMAGEIDDENIMAGEIDDENIMAGEIDDENIMAGEIDDENIMAGEIDDENIMAGE
DDENIMAGESDDENIMAGESDDENIMAGESDDENIMAGESDDENIMAGESDDENIMAGES
DENIMAGESHDENIMAGESHDENIMAGESHDENIMAGESHDENIMAGESHDENIMAGESH
ENIMAGESHIENIMAGEHIENIMAGEHIEDNIMAGEHIEDNIMAGEHIEDNIMAGEHIED
NIMAGESHIDNIMAGESIDNIMAGESIDNDIMAGESIDNDIMAGESIDNDIMAGESIDND
IMAGESHIDDIMAGESHDDIMAGESHDDIEMAGESHDIEMAGESHDIEMNAGESHDIEMN
MAGESHIDDEMAGESHIDEMAGESHIDEMNAGESHIDMNAGESHIDMNAIGESHIDMNAI
AGESHIDDENAGESHIDENAGESHIDEAGESHIDEAGESIHIDEAGESIMHIDEAGESIM
GESHIDDENIGESHIDDENIGESHIDDIGESHIDDIGESMHIDDIGESMAHIDDIGESMA
ESHIDDENIMESHIDDENIMESHIDDEMESHIDDEMESHAIDDEMESHAGIDDEMESHAG
SHIDDENIMASHIDDENIMASHIDDENASHIDDENASHIGEDDENASHIGEDDENASHIG
HIDDENIMAGHIDDENIMAGHDDENIMGHDDENIMGHDDESENIMGHDDESENIMGHDDE
IDDENIMAGEIDDENIMAGEIDENIMAGEIDENSIMAGEIDENSIMAGEIDENSIMAGEI
DDENIMAGESDDENIMAGESDENIMAGESDENIHMAGESDENIHMAGESDENIHMAGESD
DENIMAGESHDENIMAGESHDNIMAGESHDNIMIAGESHDNIMIAGESHDNIMIAGESHD
ENIMAGESHIENIMAGESHIEIMAGESHIEIMADGESHIEIMADGESHIEIMADGESHIE
NIMAGESHIDNIMAGESHIDNIMAGESHIDNIMAGESHIDNIMAGESHIDNIMAGESHID
IMAGESHIDDIMAGESHIDDIMAGESHIDDIMAGESHIDDIMAGESHIDDIMAGESHIDD
MAGESHIDDEMAGESHIDDEMAGESHIDDEMAGESHIDDEMAGESHIDDEMAGESHIDDE
AGESHIDDENAGESHIDDENAGESHIDDENAGESHIDDENAGESHIDDENAGESHIDDEN
```

Figure 3.13 shows that substitution characters do not need to be random. This is a perfectly legitimate stereogram, although it is certainly not of the random variety.

Not only does the substitution process mean that we don't need to use random letters, it means that we really don't need to use letters at all. You can use any sort of symbol to create stereograms. Experiment to find out which substitution sets are better than others.

When the substitution pattern is not random, what we've created is technically not a random dot stereogram! Actually, the term *colorfield stereogram* is often used to describe this kind of image. The `HIDDENIMAGES` substitution pattern is then referred to as the *colorfield.*

POPOUT-LITE only creates random dot stereograms, which means that it always uses some sort of random selection when picking colors for a substitution pattern. POPOUT-PRO is an example of software that will create colorfield stereograms.

Making Random Dot Substitutions

Up until now, all the example stereograms that I've presented have been made up of letters and numbers. Finally, I'm going to create an image that can really be called a random dot stereogram!

Just as letters or symbols or anything else can be used as a substitution set, so can a set of random dots. The only difference is that the resulting stereogram may be quite a bit smaller, depending on the size of the dots!

I could argue that figure 3.14 is a random dot stereogram. It just happens to be an RDS that has some really big, really strange-shaped dots. Of course, the typical stereogram that you're used to seeing will typically have much smaller dots, and the dots probably look like real dots!

With figure 3.14, I again do a simple pattern substitution using the image of 3.11. On each line, my substitution set consists of two things—a black square and a white square. Compare figure 3.14 to figure 3.12, where I had 26 letters to choose from when creating my substitution table for each line. Also compare it to figure 3.13, where I had nine unique letters to choose from (I'm not counting repeats).

In figure 3.14, I only have two symbols to pick from; that is, a black square and a white square. Still, the concept is identical. For each line, I choose a random selection of black and white squares, and make the appropriate substitutions.

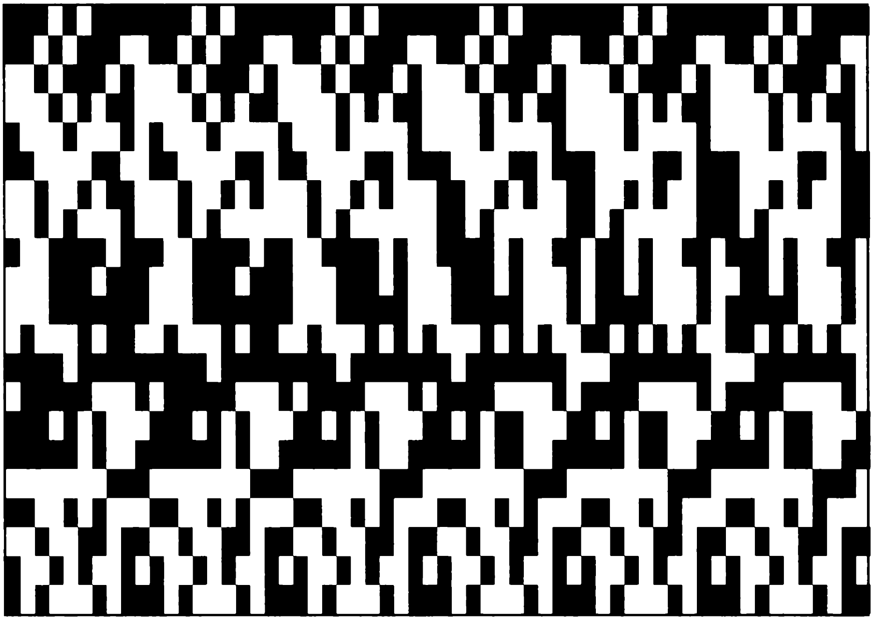
With figure 3.14, it's easy to imagine that if instead of using large black-and-white squares, I had used tiny black-and-white dots, the result would be a common black-and-white random dot stereogram! Of course, it would be a pretty small stereogram. If I had intended on using tiny dots, I would have started with a much larger image than that of figure 3.11, with a lot more than ten elements in the pattern.

Figure 3.14 was created using POPOUT-LITE. The stereogram that was created was actually a very small image, but we've enlarged it quite a bit before printing it here!

With figure 3.15, I've done just this. Figure 3.15 is (finally) an honest-to-goodness familiar looking random dot stereogram! When you think about it, it's identical in concept to all the examples presented so far, especially figure 3.14. The only difference is that I start with 80 elements in the pattern instead of just ten and proceed to use dots that are a lot smaller than the squares of figure 3.14. After the pattern is created for each line, I make a substitution using a random set of black-and-white dots. My substitution table is of the length of the maximum number of pattern elements that are used on the line, exactly as in figure 3.14.

Figure 3.14

Large random dot pattern substitution.



Of course, because figure 3.15 is a much higher resolution image, doing this by hand would be quite tedious and time consuming, simply because of the large number of dots in the image. With the computer, it takes only seconds to create this image! Figure 3.15 was also created using POPOUT-LITE.

Because of the limitations involved in printing this page, I'm only able to show you an example of a black-and-white RDS image here. You can imagine, however, that if I replaced every letter of figure 3.13 with a corresponding colored dot, I easily could create a full-color version of the image shown in figure 3.15. Color plate 2 is an example of a random dot stereogram that was created using colors other than just black and white. This image was created using four colors of dots.

Color plate 7 is another example, using only two different dot colors. These color plates were created using POPOUT-PRO in order to create the many different levels of depth, but POPOUT-LITE is also capable of creating RDS images with this kind of coloring, as well as the coloring schemes of color plates 5 and 6!

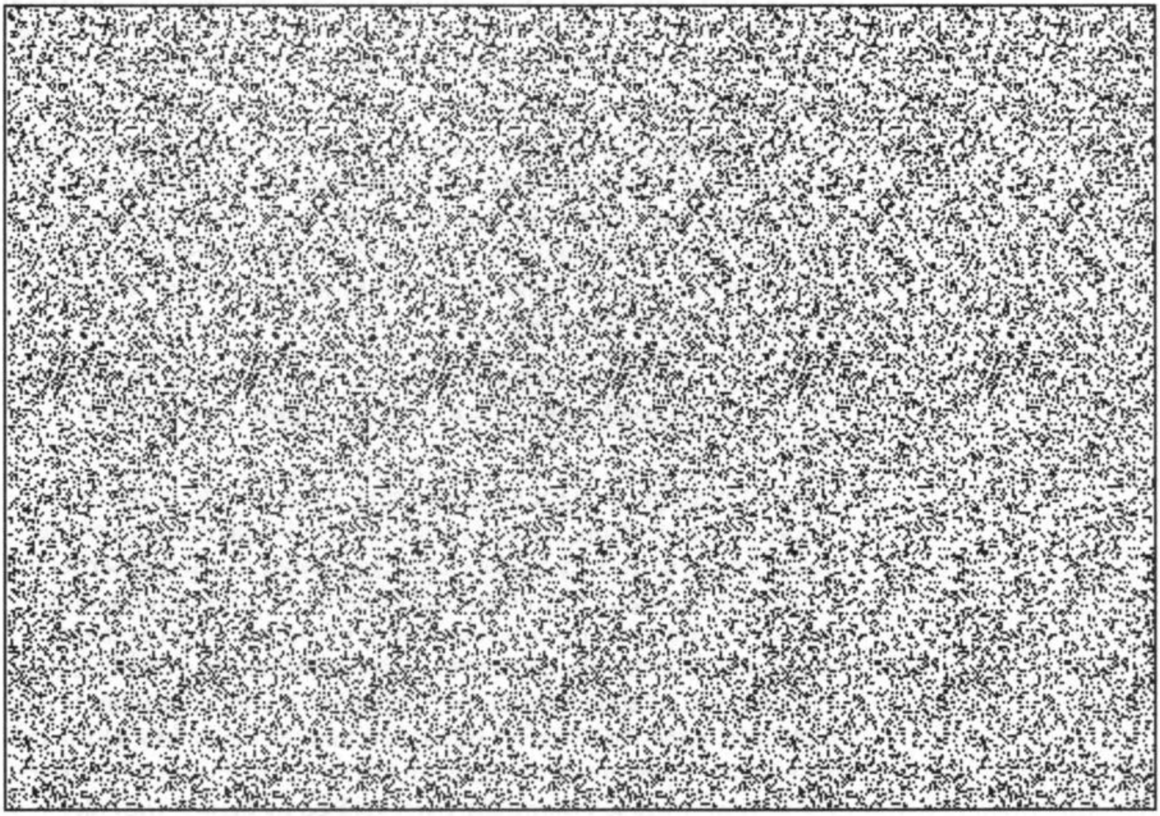


Figure 3.15

Random dot stereogram.

I could choose randomly colored dots when doing the substitution to create figure 3.15, and the result would be a randomly colored RDS image.

Making Pattern Biasing Substitutions

At this point, you have all the basics for generating random dot stereograms. From here, it's simply a matter of "souping it up" to create more interesting images.

An example of "souping it up" is to have a mechanism that lets you be a little creative when choosing the substitution set for the original pattern values. In other words, instead of just choosing random colors, or random black-and-white dots, we could "bias" the substitution set so that only certain colors were chosen.

If you really think about it, figure 3.15 is really an example of a biased selection. Rather than use all possible colors to choose from to do the substitution, I limited the computer's options to only black-and-white colored dots.

If I were creating an image containing an American flag, for example, I might want to bias the selection of substitution elements such that I only picked red, white, and blue dots. With POPOUT-LITE, you can use the Create Color menu to select exactly which colors of pixels that you want the program to use. This is done in conjunction with the Random Color RDS Output Style.

As you can see, this is a powerful concept. The biasing of the substitution pattern is what makes one RDS image appear different from another!

Another example of biasing the selection is not only to limit the colors of the selection, but to control the relative percentages of the colors used. Figure 3.15 can be considered biased in another sense. In creating the image, I instructed the computer to use a random selection of black-and-white dots such that it used three times as many white dots as it did black dots! For each line, when the computer built the substitution table for that line, it generated a set of random black-and-white dots. They weren't purely random, however, as it chose the set so that 75 percent of the dots were white, and only 25 percent of the dots were black. Color plate 2 was created by biasing the color selection so that for every four cyan dots that were generated, there were two blue dots, one black dot and one white dot also generated. This kind of biasing is what gives color plate 2 an overall bluish color.

Making Colorfield Substitutions

An example of extreme biasing when choosing the random set of dots for the pattern substitution is not to use a random set in the first place!

A large number of stereograms do not use random dots. As I mentioned earlier, these technically are not RDS images, but colorfield stereograms. Because POPOUT-LITE is an RDS image generator, it isn't capable of creating this kind of stereogram. POPOUT-PRO, on the other hand, can easily create colorfield stereograms.

Take a close look at color plates 3 and 4. These are examples of colorfield stereograms. Obviously, a pattern of some sort has been used rather than just a random selection of colored dots.

Colorfield stereograms are created by using some sort of background image, sometimes referred to as a colorfield, to obtain the substitution pattern. This background image can be some sort of graphic design, or even a scan of a photograph. As each line of the stereogram is created by the computer, a different line in the background image is used as the substitution pattern. The results of this type of stereogram can be quite amazing. Sometimes, when the height of the colorfield is the same as the height of the stereogram, the images are referred to as *wallpaper colorfields,* or *wallpaper stereograms* because the stereogram looks as if it's been created using strips of wallpaper.

Depending on the creator's choice for the background image or images used, this type of stereogram lends itself well to generating theme images. For example, a stereogram that contains a hidden American flag could use the faces of American presidents as background images.

When creating this type of stereogram, a couple of minor adjustments need to be made to the basic stereogram recipe. For example, I mentioned earlier that in generating RDS images, better results can be obtained if, after deleting an element of the pattern when creating a region of depth, that deleted element is forgotten and not used again on the line. Therefore, when inserting an element into the pattern, these elements should be new; never having been used previously on the line.

With the colorfield type of stereogram, the computer can pick from a fixed number of dots. When it comes time to do the pattern substitution, the computer must be able to pick out a dot from somewhere in the background image. Inserting new elements into the pattern can lead to trouble if there are more total elements used than there are dots in a line of the background image.

In chapter 4, I show you the basic technique that the computer uses for inserting a new element when creating a random dot stereogram. In chapter 6, I introduce some other methods that can also be used for element insertion. These other methods are better applied to images where the substitution pattern is not random.

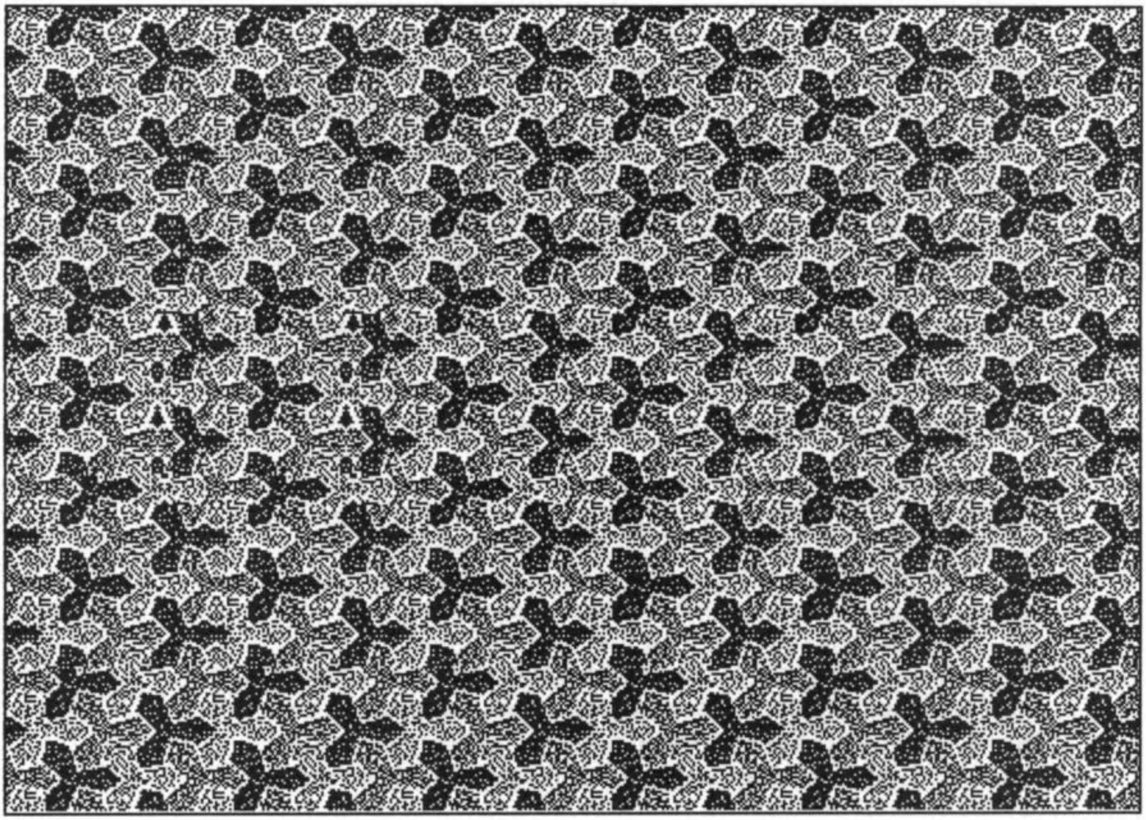


Figure 3.16

Colorfield pattern substitution.

With figure 3.16, I used POPOUT-PRO to create a colorfield stereogram using a black-and-white background image. As you can see, this is definitely *not* a random dot stereogram. There isn't a random dot anywhere in the image! It's pretty obvious that every dot was carefully selected using a second background image.

Summary

This chapter covered all the necessary ingredients for creating a basic stereogram. Because each line of a stereogram is processed one at a time, and each line is independent of any others, it's technically correct to think of each horizontal line of a stereogram as a simple but complete stereogram. The following is a list of the major steps described in this chapter:

- Start by defining a pattern of some width. This initial pattern starts with zero and counts up to the starting width minus one. This is considered to be the current pattern.

- To create a starting depth, simply write out the current pattern. When you get to the end, just repeat the pattern.

- To start or create a raised region, delete one or more elements and redefine the current pattern. The number of elements deleted will determine how much the region will be raised. Redefine the current pattern by starting at the element just after the deletion and use the remaining current pattern elements.

- To create a raised region on top of an already raised region, do the same thing. Simply delete the desired amount of elements from the current pattern and redefine the current pattern. Typically, a stereogram will have many levels of raised regions.

- To terminate a raised region, insert a new, as yet unused element into the current pattern, and then redefine the current pattern. At the point where the region will be pushed in, insert the element or elements. These inserted elements will be the first elements of the redefined pattern. The remainder of the newly redefined pattern is simply the old current pattern. Just as with the deletion, the number of element insertions will determine how much the region will be pushed in.

- Once the entire line has been processed like this, choose a random set of substitution elements. Choose as many as the highest number of pattern elements that have been used. This substitution is the key in determining how the image will look; it's color, texture, and so on.

- Using this substitution table, replace all the elements in the processed line with the corresponding element from the substitution table. You've just created one line of a stereogram.
- Because each line of a stereogram is independent of all others, the same sequence of events occurs for each line. The only difference is in the actual set of substitution elements that are chosen after the line is processed.

This chapter is intended to help you understand the specifics of how an image is created. The following chapter details the recipes, or algorithms, that a computer uses in order to create the images, and explains how a computer implements the concepts presented here.

4

Understanding Stereogram Algorithms

This book has tried to present the basic steps for creating stereograms. This chapter revisits some of those steps and presents them again in a manner more geared to the computer programmer, or to the reader interested in the step-by-step mechanisms used in stereogram generation.

After reading this chapter, you should understand the algorithms used in creating stereograms. Armed with this information, hopefully a few ambitious programmers will take these algorithms, refine them, and create a next generation of "knock-your-socks-off" stereograms!

This chapter tries not to take a language-specific approach. Many programming languages are available, and many people have varying degrees of expertise in the different languages. Rather, this chapter presents algorithms using a "pseudo-code" approach; something that can be easily understood by all readers. For example, algorithm 4.1 sums up the stereogram generation process with a pseudo-code.

Algorithm 4.1 The Really Big Picture

Step	Procedure
1	Read a line of the depth file and store it in memory.
2	Generate a line of the stereogram from this information.
3	Write the generated line to the output file on disk.
4	If there are more lines, return to step 1.

Basic Stereogram Algorithms

Some basic algorithms are used in every type of stereogram created today. The specific applications of the algorithms are different for different types of images, but the fundamentals remain the same. This section covers some of the algorithms that can be applied to all stereograms. Specifically, these algorithms are used by the POPOUT family of software.

Keep in mind the old adage, "There's more than one way to skin a cat." Show me someone who says there's only one way to create a stereogram, and I'll show you someone who understands very little about the subject! The algorithms presented in this section are mostly a review of the techniques of chapter 3, "Fundamentals of Creating a Simple Stereogram." The algorithms presented here are the most straightforward when it comes to creating a stereogram. The methods illustrated are the least math-intensive ways to skin this particular cat. You really didn't buy this book to look at a bunch of formulas anyway, did you?

Included in this chapter are those algorithms that I believe are the fundamental stereogram generation algorithms. Each is explained in detail.

This chapter isn't intended to "step on anyone's toes." By this, I mean that there has been quite a bit of study over the years in this area, and quite a lot of work has gone into creating different algorithms for stereogram generation. There is no intent to present any company's proprietary efforts, nor to infringe on anyone's patents. All the algorithms presented here were developed by the author to show a simple, obvious, and common sense approach to the problems involved in creating the images.

Maintaining the Pattern

The concept of maintaining the pattern refers back to chapter 3, where the concept of the *current pattern* is presented. A line of a stereogram is created by inserting and deleting elements into a current pattern and constantly saving that current pattern to create a final, processed pattern for that line. As discussed earlier, the pattern is dynamic, changing constantly as a line of a stereogram is created.

Because of the horizontal nature of the illusion, each line of the image is created without depending on any other line. This means that on each line, the pattern is reset to an initial sequence. As a line of the image is processed, the current pattern must be stored in the computer's memory. As raised regions are created or terminated as described in chapter 3, the current pattern must constantly be redefined.

Figure 4.1 is a copy of the text-based stereogram from chapter 3. Because this chapter refers to it occasionally, it is presented again here so that you do not have to constantly flip to chapter 3.

```
0123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345689012345689OA123456890A123456890A123456890A
```

Figure 4.1

Text-based stereogram prior to pattern substitution.

Memory requirements for storing the current pattern are not easy to calculate because they depend on the actual number of total insertions made into the pattern. In generating an RDS, new elements are inserted into the pattern each time a raised region is terminated. For example, if a region is to "drop" in depth by two units, two new elements are added to the pattern.

When a pattern width is determined, either by input from the user, input from the depth image from which a stereogram will be generated, or both, the easiest way to initialize the current pattern is to begin with the number zero and increase the numbering. The last element in the initial pattern is then the initial pattern width minus one. Figure 4.1 uses an initial pattern width of ten and initializes the ten pattern elements to 0123456789.

An important element in maintaining the pattern is to define some sort of *current element pointer*. *Pointer* is a term that simply means that it "points" to one of the elements of the current pattern. This allows you to refer to the current pattern as an entire group, and to access any one of the elements in the group using the pointer. Assume that this pointer is called "PatElemPtr," and the computer memory where the current pattern is stored is called "CurPat."

You can then refer to any element in the current pattern as the following:

```
CurPat[PatElemPtr]
```

This convention of square brackets indicates that you're interested in one element of a set of elements—in this case, one element of the current pattern. For example, if the current pattern consists of the numbers 35678901, this means that these eight numbers are stored in an area of the computer's memory called "CurPat." The first element in the pattern is referenced as CurPat[0]. In this case, CurPat[0] is a three. CurPat[1] is a five, CurPat[2] is a six, and so on.

If you assume that the initial pattern width is specified by the value "PATWIDTH," the algorithm for initializing the current pattern might look something like algorithm 4.2.

Algorithm 4.2 Pattern Initialization

Step	Procedure
1	Set PatElemPtr equal to zero.
2	Set CurPat[PatElemPtr] equal to the value of PatElemPtr.
3	Increment PatElemPtr.
4	If PatElemPtr is less than PATWIDTH, go to step 2.
5	Set PatElemPtr equal to zero.
6	Set CurPatWidth equal to PATWIDTH.
7	Set MaxPatWidth equal to PATWIDTH.

At this point, you're ready to process an image. The current pattern is initialized, and a pointer is pointing to the first (the 0th element) in the pattern.

As the image is processed, the value of PatElemPtr is incremented for each pixel in the line. Remember, all lines are independent, so this initialization takes place before processing each line. When PatElemPtr reaches the total number of elements in the pattern (which is specified by the CurPatWidth variable), it is reset to zero. Doing this enables you to cycle through the pattern elements over and over again as the line is processed. The result is that the pattern repeats.

You may have noticed that algorithm 4.2 uses a couple of other variables that I haven't mentioned yet. You will see that these are useful later. CurPatWidth is a variable specifying the length of the current pattern. Because CurPatWidth is initialized to have PATWIDTH elements, I simply set it equal to PATWIDTH. The other variable, called "MaxPatWidth," is used to save a little time later. It specifies the maximum number of elements ever used in processing the line. You use this later in doing pattern substitution.

To put this into a slightly larger perspective, assume that you happen to be working on a line where you won't be making any modifications to the pattern. A good example of this is the top and bottom few lines of figure 4.1. The algorithm for "processing" this *no-depth line* is illustrated in algorithm 4.3.

I've introduced a couple of new things into algorithm 4.3 that you haven't seen yet. One is something called "WIDTH." I capitalized it to show that this is a constant value, and refers to the length of the line. When the line is processed, we have to know when we're done processing. Knowing how wide the line is will help in determining when we're finished. The other thing is a variable called "CurrentX," which is how I keep track of how far along I am in processing the line. If I start out with CurrentX equal to zero, and then increment it by one after processing each element of the line, I can compare it to the WIDTH value to see if I've finished.

Algorithm 4.3 Basic "Single Level" Processing Loop

Step	Procedure
1	Initialize the Pattern using algorithm 4.2.
2	Set the variable CurrentX to zero.
3	Save the value CurPat[PatElemPtr].
4	Increment PatElemPtr.
5	If PatElemPtr equals CurPatWidth, Set PatElemPtr to zero.
6	Increment CurrentX.
7	If CurrentX is less than WIDTH, go to step 3.

Note the use of a variable called "CurrentX," which simply keeps track of the number of pixels processed. If WIDTH is the width of the stereogram, CurrentX is how you know when you're finished processing the line.

Algorithm 4.3 is the basic "loop" of processing a line to create a stereogram. With a ten-element initial pattern and a WIDTH of 60, this algorithm generates the following line:

```
012345678901234567890123456789012345678901234567890123456789
```

Because I made no modifications to the starting pattern, no depth is impressed into this line.

Reading a Depth File

The *depth file,* or *depth image,* is the image that the stereogram software uses to get its depth information when creating an image. This image is created by the user. In practice, creation of the depth image accounts for the vast majority of time involved in creating a stereogram! For most PC users, the depth image is typically created as a drawing using some sort of paint program. In most stereogram generators, the depth file comes in the form of a grayscale image. The common convention is that a very dark, or black, pixel in the depth image means that the corresponding pixel in the stereogram will be the farthest from the observer. The brighter the pixels in the depth image, the closer the apparent distance between the object and the observer. The whitest pixels in the depth file are translated to the points in the stereogram closest to the observer.

Part 2 of this book explains how to generate depth images and convert them to random dot stereograms. The quality of the depth image is a very big determining factor in the quality of the final stereogram. This "black is far, white is close" concept is a convention, but by no means the only way to create a depth image. For example, figure 4.2 is a copy of the depth image used to create the stereogram of figure 4.1. Instead of a graphic image consisting of grayscales, I simply used a text file containing numbers. If you compare this to figure 4.1, the zeroes in the depth image generate pixels (characters) in the stereogram that are farther from the observer than the ones, and the twos are the closest of all. If you were to translate figure 4.2 to a conventional depth image, the zeroes would be black pixels; the twos would be white; and the ones would be a shade of gray halfway between black and white.

An excellent example of where this "black is far, white is close" convention is not always practical is with the POPOUT-LITE software. Since POPOUT-LITE was specifically designed to be used with Paintbrush, the convention had to be altered a bit, simply because Paintbrush isn't capable of creating more than two different shades of gray. In fact, Paintbrush can only save a total of 16 unique colors, and you don't even have the option of choosing what those colors are! For this reason, we adapted POPOUT-LITE to the limitations of Paintbrush, and used the 16 different colors that Paintbrush *can* create to represent the different levels of depth.

Because POPOUT-PRO was designed as a more powerful program to be used with something *other* than Paintbrush, it honors the grayscale convention described above. Most of the images in the book that contain more than 16 levels were created with POPOUT-PRO, using a grayscale depth image.

Figure 4.2

Simple text-based depth image.

```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000001111111111110000000000000000000000000000000000000000
0000000000001111111111110000000000000000000000000000000000000000
0000000000001111111111110000000000000111111111111000000000000000
0000000000001111111111110000000000000111111111111000000000000000
0000000000001111111111222222222222222221111111111000000000000000
0000000000000000000000222222222222222221111111111000000000000000
0000000000000000000000222222222222222221111111111000000000000000
0000000000000000000000222222222222222220000000000000000000000000
0000000000000000111111222222222222222220000000000000000000000000
0000000000000000111111111111000000000000000000000000000000000000
0000000000000000111111111111000000000000000000000000000000000000
0000000000000000111111111111000000000000000000000000000000000000
0000000000000000111111111111000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
```

Although figure 4.2 works fine as a depth image for a simple, low resolution text-based stereogram, it's not really suitable for use in conventional stereograms, which are large, high-resolution images. Figure 4.3 is a more conventional depth image.

Because a stereogram is generally processed a line at a time, a depth image is typically read a line at a time. After a line of the depth image is read, the numerical values of the individual pixels determine whether a pattern element is to be deleted, inserted, or left alone.

A simple way to look at this is with a short algorithmic example. Assume that you read a line of the depth image into the computer and stored it in a piece of memory called "DepthLine." Furthermore, assume that the image is a 256-level grayscale image, where the individual pixels have a value of 0 for black and 255 for white. The decision-making process of what to do regarding the pattern could look like algorithm 4.4. The basic idea of algorithm 4.4 is to scan across the length of a line of the depth image, all the while comparing the grayscale value of the pixel with the grayscale value of the previous pixel. You'll notice a couple of variables called "LastPixel" and "CurrentPixel." These variables are updated to hold the grayscale value of the previous pixel that was examined, and the grayscale value of the pixel currently being examined. By comparing these two values, a decision is made whether to insert or delete an element or elements into (or from) the current pattern, or else to leave it alone and continue with the next pixel.

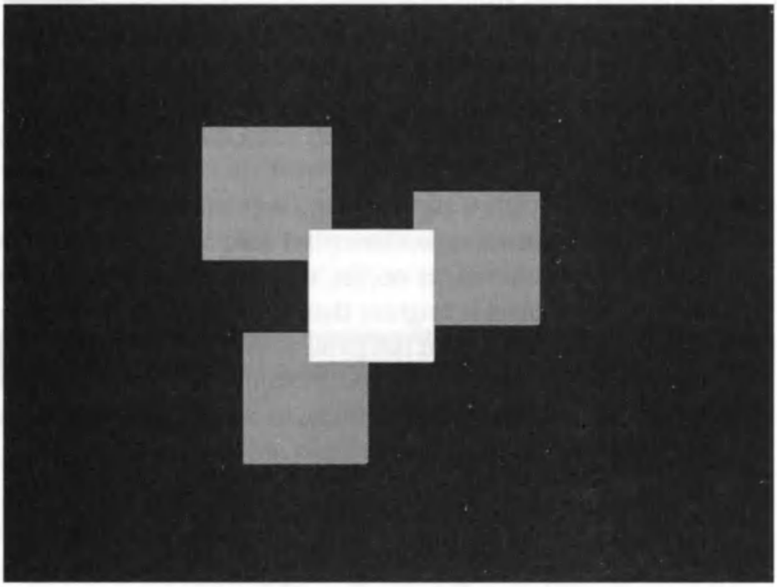


Figure 4.3

Simple depth image.

Algorithm 4.4 "Big Picture" of Processing a Line

Step	Procedure
1	Read a line of the depth image to DepthLine.
2	Set CurrentX equal to zero.
3	Set LastPixel equal to DepthLine[0].
4	Set CurrentPixel equal to DepthLine[CurrentX].

continues

Algorithm 4.4 Continued

Step	Procedure
5	If CurrentPixel is greater than LastPixel, Delete (CurrentPixel – LastPixel) elements from the current pattern.
6	If CurrentPixel is less than LastPixel, Insert (LastPixel – CurrentPixel) elements into the current pattern.
7	Otherwise, leave the current pattern alone.
8	Increment CurrentX.
9	Set LastPixel equal to CurrentPixel.
10	If CurrentX is less than WIDTH, go to step 4.

As you can see, the algorithm compares the current pixel in the depth image with the previous pixel read from the depth image. If no change in pixel intensity occurs, no change is made to the pattern. If the current pixel is brighter than the last pixel, an element or elements is deleted from the pattern to create a region of depth. If the current pixel is darker than the previous pixel, an element or elements is inserted into the current pattern to terminate a raised region.

In general, the difference between the brightness of the current pixel and the previous pixel is not always zero or one. In the simplest implementation of the algorithm, this difference value is used to determine how many elements need to be deleted or inserted into the pattern. Later you see that with some stereogram software, this may not necessarily be true.

Deleting Elements from the Pattern

As you saw earlier, creating a region of depth is done by deleting one or more elements from the current pattern and then redefining the pattern. The more elements deleted, the closer the raised region appears to the observer.

Deleting an element from the current pattern isn't all that difficult. The technique is pretty straightforward—just a little tricky at first.

Notice that steps 5, 6 and 7 of algorithm 4.4 is a decision-making process. A decision is made whether to insert or delete an element or elements, or just to leave the current pattern alone. When the decision is made to delete an element from the pattern (which will create a raised region), it is typically done so by a process like that of algorithm 4.4, where the grayscale pixel intensities of the depth image are examined. As the preceding algorithm shows, an increase in depth file pixel intensity between the current pixel and the last pixel means that an element or elements must be deleted from the current pattern.

Consider algorithm 4.5. I've introduced some new variables in this algorithm. TempPat is just a section of the computer's memory that I'm going to use as a temporary holding area to store the current pattern. NewPtr is a pointer, or index, into the new, redefined current pattern. NewWidth is the new width of the current pattern after the element or elements are deleted, and NUMDEL is a value that is passed to algorithm 4.5, telling it just how many elements to delete. Notice that if you compare this with step 5 of algorithm 4.4, NUMDEL will have the value of CurrentPixel minus LastPixel.

Algorithm 4.5 Pattern Element Deletion

Step	Procedure
1	Copy all the elements in CurPat to TempPat.
2	Set NewPtr to zero.
3	Set NewWidth to CurPatWidth – NUMDEL.
4	Add NUMDEL to CurElemPtr, modulus CurPatWidth.
5	Copy TempPat[CurElemPtr] to CurPat[NewPtr].
6	Increment NewPtr.
7	Increment CurElemPtr.
8	If CurElemPtr equals CurPatWidth, set CurElemPtr to zero.
9	If NewPtr is less than NewWidth, go to step 5.
10	Set CurPatWidth to NewWidth.
11	Set PatElemPtr to zero.

Let's examine what algorithm 4.5 does. Let's assume that before the algorithm is started, that a value equal to CurrentPixel minus LastPixel has been assigned to NUMDEL. As I mentioned earlier, this will tell the algorithm the number of elements that should be deleted.

Algorithm 4.5 begins by copying all current pattern elements to a temporary area called "TempPat." This is done because at least some of them now will be copied back into CurPat. Do not copy back the elements you want to delete. To make life easy, "redefine" the current pattern as discussed in chapter 2, "The Basic Concepts." That way, you don't have to worry about it later.

Step 2 uses a variable called "NewPtr," which is set to zero. This is used as an index for pointing into CurPat when the elements are copied from the temporary storage.

Step 3 creates a variable called "NewWidth," which will be the new current pattern width after the deletions are complete. NewWidth is simply the old current pattern value minus the number of elements deleted.

Step 4 is tricky if you're not familiar with modulus arithmetic. This adds the number of elements to delete to the value of the current element pointer, making sure that the result isn't any bigger than the number of elements in the pattern. If the algorithm is deleting one element, that element should be the element pointed to by the current element pointer. If the algorithm is deleting two elements, they should be the current element and the next one in the current pattern. Step 4 increments the current pointer past the elements to be deleted. The modulus arithmetic "wraps" CurElemPtr back around to make sure that CurElemPtr doesn't point past the last element in the pattern specified by CurPatWidth.

Step 5 copies the pattern from the temporary area back into the pattern storage CurPat, and at the same time redefines the current pattern. The first element to be copied is the element just past the deleted ones. When you're all finished with this, CurPat[0] contains the first element in the redefined current pattern.

Steps 6 through 10 "loop" until all remaining elements are put back into CurPat at their newly assigned, redefined positions.

Step 11 updates the current pattern width to reflect the deletion; and step 12 completes the redefinition by moving the current element to the beginning of the redefined pattern.

Take the time to walk through algorithm 4.5 because it is very important. It is the basic algorithm for deleting pattern elements, and thereby creating raised regions when this newly redefined pattern is used.

Inserting Elements into the Pattern

Just as deleting elements from the pattern creates raised regions, inserting new elements into the pattern causes them to be *terminated.* The algorithm for inserting new elements is similar. The only real difference is in deciding from where the new, inserted elements are going to come.

Stereogram generation uses a couple of different element insertion algorithms. One uses the philosophy that if an element is inserted, then it must be a new, previously unused element. The other is that if elements are to be inserted, then you should use elements that may have been previously deleted.

Both of these concepts are valid, and both have their place in creating stereograms. It turns out from experimentation that in generating stereograms of the random dot variety, better images are often produced when never-before-used elements are introduced into the pattern.

NOTE: Recall from chapter 2 that in the final phase of creating a stereogram, a substitution pattern is selected. This substitution pattern can be selected either randomly, or from a slice or line of some other "colorfield image." If the substitution pattern is random, the result will be a random dot stereogram. Otherwise, the result will be a colorfield stereogram. Knowing this in advance will help decide just how pattern element insertions should be performed. POPOUT-LITE doesn't have to worry about this decision of where the substitution pattern will come from because it always creates random dot stereograms. POPOUT-PRO, on the other hand, is an example of a program that needs to make this decision.

When creating stereograms using a colorfield background image, you can't just create new elements forever. Sooner or later, those pattern elements will need to be replaced with a line from the colorfield image. Because the width of the colorfield is fixed, the total possible number of used pattern elements must also be fixed.

For now, this section addresses the case of the random dot stereogram and discusses the concept of inserting brand new elements. In chapter 6, I show you some of the other techniques that are used for inserting pattern elements.

In doing this, one extra value must be considered—the total number of new elements inserted into the pattern. When you finish processing an entire line, the actual number of elements in the current pattern width at the end may not reflect the total number of new elements introduced.

The reason for this is that sometime after the insertion, some deletions may have occurred. It is possible, for example, to begin processing a line with an initial pattern width of, say, ten elements, and end up processing also with ten elements in the current pattern. Figure 4.1 shows this exact thing. The current pattern width was ten at the end of every line. Notice, however, that in any line that had raised regions, the ten elements of the ending pattern were different from the starting pattern of zero through nine. In fact, it is possible, depending on the number of, depth of, and placement of regions, that the ending pattern may have none of the original elements, even though it may be the exact same length as the starting pattern.

As with the element deletion algorithm (4.5), algorithm 4.6 combines the functions of element insertion and pattern redefinition. It assumes that a value NUMINS is used to specify to the algorithm how many new elements are to be inserted. The basic RDS element insertion algorithm is stated in algorithm 4.6.

Algorithm 4.6 Pattern Element Insertion with New Elements

Step	Procedure
1	Copy all the elements in CurPat to TempPat.
2	Set NewPtr to zero.
3	Set Count to zero.

Step	Procedure
4	Set CurPat[NewPtr] to (MaxPatWidth + Count).
5	Increment NewPtr.
6	Increment Count.
7	If Count is less than NUMINS, go to step 4.
8	Set NewWidth to CurPatWidth + NUMINS.
9	Copy TempPat[CurElemPtr] to CurPat[NewPtr].
10	Increment NewPtr.
11	Increment CurElemPtr.
12	If CurElemPtr equals CurPatWidth, set CurElemPtr to zero.
13	If NewPtr is less than NewWidth, go to step 5.
14	Set CurPatWidth to NewWidth.
15	Set PatElemPtr to zero.
16	Add NUMINS to MaxPatWidth.

Now take a closer look at the steps of algorithm 4.6. Algorithm 4.6 begins by copying all the pattern elements to a temporary area. After they are tucked away, the algorithm performs the insertion. If you recall from chapter 3, new elements are inserted into the pattern first, so that the new elements are the first ones of the redefined pattern. The algorithm does just this.

As with the delete algorithm, NewPtr is initialized to zero and is used as a pointer when copying the elements back from the temporary storage. First, however, NewPtr is used in writing the new elements into CurPat.

Step 3 initializes a variable called Count, which is used to cycle through the insertions.

Step 4 is a bit tricky to understand at first, but is the key in creating new, unused pattern elements. When the algorithm to initialize the pattern was first created, the variable MaxPatWidth was set equal to PATWIDTH, the width of the original, starting pattern. I mentioned that it would be used later; now is the time.

MaxPatWidth is updated by the element insert algorithm; its purpose is to indicate the total number of elements inserted into the pattern, including the original ones. Because elements of the pattern are just numbers and the original ones started with zero and just kept counting, MaxPatWidth tells you the next number "in line" to be put into the pattern, enabling the algorithm to insert new, never-before-used numbers.

Step 4 actually comes up with a never-before-used number. By adding Count to MaxPatWidth and incrementing Count (see step 6) each time, the algorithm keeps putting never-before-used numbers into the beginning of the pattern for as many times as there are elements to be inserted.

Steps 5 through 7 complete this loop, inserting new, sequential numbers into the beginning of the pattern.

Step 8 updates NewWidth to reflect the new size of the pattern after the new elements are inserted.

Steps 9 through 13 make up the loop where all the original pattern elements (before the insertion) are copied back into CurPat, just after the new, inserted elements.

Step 14 updates the current pattern width to reflect the insertion of the new elements, and step 15 completes the pattern redefinition by declaring CurPat[0] to be the first element of the current pattern at this point in the image.

Finally, step 16 updates MaxPatWidth to reflect the maximum elements used so far. This way, the next time the algorithm is used to do another pattern insertion, it can again use previously unused values for the inserted pattern elements.

As with element deletion, this algorithm is very important in generating stereograms. Take a few minutes to review the algorithm to make sure that you understand the concept.

Algorithms 4.5 and 4.6, the element delete and element insert algorithms, can replace the words "Delete" and "Insert" back in steps 5 and 6 of algorithm 4.4. If you add algorithm 4.2 to the beginning of 4.4, you then have the entire recipe for creating a pattern for one line of a stereogram.

When you get right down to it, all you've been doing so far is fiddling with creating a final pattern by inserting and deleting elements into and from a current pattern, and using that constantly changing current pattern to create the final pattern. If you continue to do this for each line, the result is something like figure 4.1.

Although figure 4.1 is technically a stereogram—and it certainly makes for a nice tool in illustrating some of these concepts—it's really not what you'd call a "finished product." As you saw in chapter 3, a final step of pattern substitution must be performed to actually create a presentable image.

Basic Pattern Substitution

The key to turning a processed pattern into a presentable stereogram is in substituting the pattern elements with something a bit more interesting than pattern element numbers. As you learned in chapter 3, this can be as simple as a set of characters from the alphabet. Of course, the higher the resolution of the stereogram, the smaller the individual elements become. With most stereograms, dots, or pixels, are substituted for the pattern elements.

The line-independence quality of stereograms can make pattern substitution easy for some types of images and a bit more difficult for others. For example, in generating a random dot stereogram, each completed pattern line is simply replaced with a set of random dots. This completed line can be written to disk before the next line is even started. It's all relatively easy. For colorfield images, the stereogram software must be a bit more intelligent as it must fetch a specific slice, or line, from the colorfield image to use as the substitution data. The image can still be processed one line at a time; it just takes a bit more sophistication to implement it.

For basic RDS-type substitution, you first have to decide on a set of pixels, characters, colors, symbols, or whatever, to use as your substitution set.

The most important parameter passed to the pattern substitution algorithm is the MaxPatWidth value used in the algorithms presented so far. If you recall, this is the number initialized to equal the initial pattern width before the line is processed. As the line is processed, inserting elements into the pattern causes this value to be updated. This updated MaxPatWidth reflects the total number of elements used.

When an entire line is processed, the value of MaxPatWidth is the total number of substitution elements that must be utilized. In figure 4.1, for example, the value of MaxPatWidth after processing the first line would be 10 because no modifications were made to the original pattern. As you can see, only the 10 elements, zero through nine, appear anywhere on the top line. Farther down in the image are lines where a couple of pattern element insertions have been made, and the characters "A" and "B" are used to represent the pattern values of 10 and 11, respectively. On a line like this, MaxPatWidth would have a value of 12 after the line is processed. This makes sense because a total of 12 different pattern element values must be substituted.

One thing that really separates high-quality stereograms from those of lesser quality is the choice of substitution values. As mentioned in chapter 3, choosing random values can, and usually is, biased in one sense or another. For example, for a simple black-and-white stereogram, using a purely statistically random set of black-and-white pixels as the substitution set usually results in very dark images when sent to the printer. This is especially noticeable in high-resolution images. The choice is sometimes biased such that the user can control the percentage of white versus black pixels chosen for the substitution set.

Another example is using random colors as the substitution set. Some computers have the capability of displaying over 16 million different colors. If you choose a statistically random selection from these 16 million plus colors, the resulting image really isn't all that interesting. Actually, it looks quite dull. So many of those colors are dark, dim, and downright uninteresting, and Murphy's Law says that you're bound to get the ugliest ones of the bunch. A better approach is to bias the selection of the substitution set by limiting the selection to a subset of the colors.

For example, suppose that you are creating a stereogram containing the image of a heart with an arrow through it and the words "I love Karen" written across it. Karen might well be pleased with any color combination while viewing her secret message, but may really enjoy the image if it were done by just choosing different shades of pink or blue dots, or whatever her favorite color happens to be.

Selecting a set of pattern element substitution values for a random dot image does not have to be that difficult, but there should be some sort of mechanism to "filter out" all but a selected set of values. Algorithm 4.7 does something like this.

Algorithm 4.7 Selecting a Pattern Element Substitution Set

Step	Procedure
1	Set Count equal to zero.
2	Fetch a random value.
3	If it's not a good choice, go to step 2.
4	Store the random value in SubstVals[Count].
5	Increment Count.
6	If Count is less than MaxPatWidth, go to step 2.

As you can see, algorithm 4.7 introduces a storage area called "SubstVals," which will contain the values that will be used to create the final image. SubstVals must be able to store as many values as the size of MaxPatWidth.

The algorithm also uses a variable called "Count," which loops until the algorithm collects all the values it needs. Step 6 keeps it looping until it fetches as many "good" values as it needs to replace all the pattern elements on the line.

Algorithm 4.7 is very general and definitely not the best way to do this for all cases. Step 3 was purposely left as a very general statement so that this could conceivably work in all cases.

POPOUT-LITE makes the decision of how to select colors for the substitution pattern in a number of different ways. If you've chosen the Black & White Output Style, the decision is easy. All the dots that are used are either black or white! If the Output Style is selected as Random Color, then POPOUT-LITE looks to see if you've biased the selection by way of the Create Color menu. If you haven't, it will choose randomly from 16 different colors. If you have, it will choose the dot colors according to your selection. With the Custom Color Output Style, POPOUT-LITE makes its decisions regarding dot color by looking at the "Color Definition File" that you've created. While biasing a color selection and creating a color file will be discussed in detail later, for now I want you to understand that these are practical examples of just how this selection of the substitution pattern takes place.

When a random set of substitution values is selected, you're ready to create an image. Given the variables used in algorithm 4.7, algorithm 4.8 shows how these values are used. Assume that a pattern has been created for a line similar to one of the lines of figure 4.1, and assume that this line of pattern elements is stored in an area of memory called "ElementLine."

Algorithm 4.8 Pattern Substitution

Step	Procedure
1	Set CountX equal to zero.
2	Set PatElement equal to ElementLine[CountX}.
3	Set OutputPixel equal to SubstVals[PatElement].
4	Save OutputPixel.
5	Increment CountX.
6	If CountX is less than WIDTH, go to step 2.

Algorithm 4.8 actually creates pixels that will be written to disk as the stereogram image. Step 1 introduces a variable called "CountX," which is used both to index into the ElementLine to fetch individual values, and also as a looping variable.

Step 2 fetches the pattern element at the current position.

Step 3 actually does the substitution. It replaces the pattern element value with the substitution pixel value. If you were to replace step 3 with something like the following:

Step 3. Set OutputPixel equal to PatElement

the algorithm would write the actual pattern element as the output pixel. This is essentially what I did to create the image of figure 4.1. Up until now, you were probably impressed, thinking that I had done it by hand instead of letting the computer create it!

Step 4 saves the pixel because it is now part of the final image.

Step 5 increments CountX to get ready for the next time through the loop, and step 6 keeps things going until the algorithm substitutes and saves all the pixels on a line.

A very important concept is that algorithm 4.8 is used to create a single line of the image. For each line, a new set of random values must be chosen and stored into SubstVals. If not, the resulting stereogram doesn't look very random. In fact, it will remind you very much of figure 4.1.

Putting It All Together

Between the basic concepts presented in chapter 3, as well as the algorithms presented in this chapter, you really have everything you need either to understand how a basic stereogram is created, or to conceivably create one manually using graph paper, a pencil, and a great deal of patience!

One last algorithm, 4.9, uses most of the algorithms already presented. This shows the "big picture" of creating an entire image and hopefully ties up some loose ends.

Algorithm 4.9 Generating a Stereogram

Step	Procedure
1	Set CountY equal to zero.
2	Read one line of the depth file to DepthLine.
3	Initialize the Pattern (algorithm 4.2)
4	Set CurrentX equal to zero.
5	Set LastPixel equal to DepthLine[0].
6	Set CurrentPixel equal to DepthLine[CurrentX].
7	If CurrentPixel is greater than LastPixel, Delete (CurrentPixel – LastPixel) elements (algorithm 4.5).
8	If CurrentPixel is less than LastPixel, Insert (LastPixel – CurrentPixel) elements (algorithm 4.6).
9	Increment PatElemPtr (per algorithm 4.3).

continues

Algorithm 4.9 Continued

Step	Procedure
10	If PatElemPtr equals CurPatWidth, Set PatElemPtr to zero.
11	Increment CurrentX.
12	Set LastPixel equal to CurrentPixel.
13	If CurrentX is less than WIDTH, go to step 4.
14	Perform a Pattern Substitution (algorithm 4.8).
15	Write the line to disk.
16	Increment CountY.
17	If CountY is less than HEIGHT, go to step 2.

Algorithm 4.9 basically illustrates all the essential steps to creating a stereogram. Rather than type out all the little steps for inserting elements, deleting elements, and so on, algorithm 4.9 refers back to the actual algorithm that does this function. This prevents algorithm 4.9 from becoming too cluttered. Computer programmers will recognize these as subroutine calls.

Some holes exist, as some readers may notice. For example, this algorithm is geared toward the generation of a simple RDS. As mentioned earlier, the element insertion algorithm (algorithm 4.6) is usually done slightly differently when generating a colorfield image.

Also, another concept is sometimes used with certain types of colorfield stereograms that aren't addressed here. For some images, you can obtain better results by processing each line of the image starting not from an edge, but from the middle. One half the line is processed by starting at the middle and moving to the right; and the other half is processed by starting again at the middle and then moving toward the left. You may recall from chapter 2 that instead of processing an image from left to right, you can just as easily process it from right to left. This concept comes in very handy for certain types of images.

These, as well as other more advanced algorithms and topics are discussed in chapter 6, which is dedicated to some of these more advanced subjects.

Summary

Whether a stereogram is created using a computer or generated by a dedicated (and extremely patient) individual drawing dots on a piece of paper, you must follow a step-by-step process. These recipes, or algorithms, were presented in this chapter in such a way as to show all the necessary steps required in the generation of a viewable random dot stereogram.

Computer programmers can translate these algorithms into actual computer software for generating the images. For the rest of us, they simply give some insight into the details of how the illusion is actually generated.

The following list describes the algorithms that were presented in this chapter:

- Algorithm 4.1 shows how a stereogram is created, viewed from a very high level. Not having much detail, I simply want you to realize the image is entirely processed just one line at a time.
- In algorithm 4.2, you saw how the current pattern is initialized before beginning to process a line. Given a desired initial pattern width, the algorithm prepares for processing by defining the current pattern.
- Processing a line of the stereogram consists of building a final pattern by constantly repeating the current pattern, whatever it happens to be. Algorithm 4.3 shows an example of creating a line that contains no changes in depth, where the result ends up being simply a long repetition of the starting pattern.
- Typically, a stereogram will be created where there are changes of apparent depth within a line. Regions of depth must be created and terminated. Usually, this is done by something like algorithm 4.4, which reads a depth image and makes the decisions of whether raised regions are to be created or terminated. The depth image contains the stereogram's "hidden image." It is impressed into the stereogram by virtue of the fact that algorithm 4.4 decides how the pattern is to be altered by examining the grayscale pixel values of the depth image.

- Algorithm 4.5 shows how an arbitrary number of elements are deleted from the current pattern. This arbitrary number is determined by algorithm 4.4, which in turn directs algorithm 4.5 to delete the required number of elements. Doing so will create a region of depth. The apparent change in depth is determined by the number of elements that will be deleted. Once the elements are deleted, algorithm 4.5 redefines the current pattern so that processing may continue.

- When algorithm 4.4 decides that a region of depth must be terminated, it uses algorithm 4.6 to do this. Algorithm 4.6 is used to insert new elements into the current pattern, and then used to redefine the current pattern as beginning with the newly inserted elements. There are a number of different element insertion algorithms, and algorithm 4.6 works best when creating random dot stereograms rather than colorfield stereograms.

- After a line has been processed and a final pattern for the line has been created, algorithm 4.7 is used to select a substitution pattern. In the final step of processing a line, the substitution pattern will replace the existing pattern. The type of stereogram (random dot or colorfield), as well as the color, appearance and quality of the image, will in a large part be determined by just how this substitution pattern is selected.

- Algorithm 4.8 is the process for actually doing this pattern substitution, once algorithm 4.7 has chosen the substitution pattern to be used. It performs a simple one-to-one mapping, replacing the elements that make up the final pattern with corresponding elements from the substitution pattern.

- Finally, you were again given a higher level view of the entire process in the form of algorithm 4.9. This algorithm is the "top level" process that ties all of the other ones together. In short, algorithm 4.9 is the basic recipe for creating a stereogram.

5

Stereogram Generation Parameters

Every stereogram, regardless of its complexity or how it was created, includes within it the results of processing within the limits of a set of parametric values. Sound confusing? It really isn't.

When you turn on your television, you see an image generated and presented to you. Depending on individual tastes, you may not be perfectly satisfied with the quality of the image in front of you. Fortunately, most television manufacturers give you a set of knobs or switches that enable you to fiddle with a set of adjustments built into the TV.

You can adjust the volume, brightness, contrast, tint, or any number of other controls, depending on the particular model. Now think of all these different television adjustments as *parameters*— entities that affect the electronic innards of the television in such a way as to change the resulting picture and sound. Unless it's a very badly designed television, you don't need to be a television engineer or technician to adjust a picture to your liking. You simply need to know what you like.

Stereogram generation works the same way. A stereogram is an optical illusion, and as optical illusions go, a stereogram is pretty far up there in the Complicated category. By far, most stereograms are generated by computer, not by hand. Typically, stereogram generation software provides users a set of "knobs," enabling them to adjust the appearance and quality of the final stereogram.

Some software is more sophisticated than other software in this department, but then some TVs have more knobs and switches than others. A television that has no contrast adjustment, for example, still must generate a picture with contrasting colors or grayscales. It just means that this "parameter" has been "hardwired" into the design of the television, instead of allowing the user to adjust it. The same thing goes with stereogram software. Some packages have "hardwired" (or more accurately, "hardcoded") parameters.

This chapter introduces you to some of the parameters, or adjustments, that affect the appearance and quality of a stereogram. A number of parameters are discussed, some of which you may not have seen in stereogram software. Others may exist in your software, but may be labeled with different names. Still, the function exists, and plays a part in creating the stereogram.

Some stereogram generation parameters are more essential than others. For example, although not all televisions enable you to fiddle with the bass and treble of the audio, virtually all give you the capability to adjust the volume. In stereogram generation, the same concepts apply. An example of an essential parameter is the user being able to specify the value of the initial pattern width.

Stereogram software programmers have yet to agree on a universal language with regards to stereogram terminology. Readers familiar with multiple stereogram-generating software packages have found this to be very true, even to the point of being annoying, if not totally confusing. Some commonly used stereogram-related terms have even been trademarked by different companies, so that they can't be freely used by everyone. Even the word "stereogram" is just one of many terms used to describe the illusion.

This chapter attempts to present these parameters using the most common terminology possible, interspersed with terminology that is at least descriptive of the parameter being discussed. The more common parameters are discussed first, followed by the more obscure or more complex.

Pattern Width

Just about all stereogram programs offer you a chance to specify the initial pattern width. POPOUT-LITE certainly allows you to do so. Depending on the particular program, this parameter may be available under a number of different names. Field width, strip width, and colorfield width are just a few. After you become familiar with your software, you quickly find the parameter that, for consistency, this chapter refers to as the *pattern width*.

As you saw in chapters 2 and 3, the pattern width is an essential element in creating an image. Usually, the pattern width parameter is specified in units of pixels, or dots, on the final stereogram.

Certain types of stereograms have no need for a pattern width parameter. For example, if you're generating a colorfield type of image, the starting pattern width is usually implied as being the width of the colorfield image. It's important for you to realize this. When creating this type of image, just think of the width (in pixels) of the colorfield image as the initial pattern width value. Then, all the following discussions apply.

The pattern width has the following two basic functions:

- The pattern width affects how easy it is to view your stereogram. This can make or break your image.
- The pattern width also, to some extent, affects the amount of apparent depth in the image.

Perhaps the most important aspect of the pattern width is in translating the parameter into something "real," such as inches or centimeters. As mentioned earlier, the pattern width parameter of most stereogram generating software is specified in pixels, rather than in a linear measure such as inches or centimeters.

Because *pixels* is an arbitrary term that has no real dimensions, it's important for you to be able to translate pixels into linear measure. This implies that you know something about the medium on which you display your image. For example, you may be printing your stereograms to a 300 dpi printer, which means that the printer is capable of printing 300 dots in the span of one inch. If every one of these dots is a pixel, you potentially could generate a stereogram with 300 pixels per inch.

Your PC monitor, on the other hand, is a much lower resolution device. Depending on your computer's graphic capability, as well as the physical size of your monitor, your screen resolution may be very different than someone else's. In creating stereograms, it's very helpful to know the resolution of your monitor.

> **NOTE:** Using a ruler, measure the distance (in inches) from the left-hand side of the viewing area of your monitor to the right-hand side. Just measure the distance where pixels are actually displayed. Call that value "D." Next, you need to know how many pixels are displayed on one line of your PC's monitor. Call this number "P." If you're not sure what this number is, you may want to refer to your computer's manual. In your manual, it is probably referred to as the resolution of your monitor or graphics adapter. This is most likely a number like 640, 800, 1020, or 1280. Divide D into P. The result is the resolution of your monitor in pixels per inch. For example, a lot of computers are set up with a screen resolution of "800 x 600". This means that there are 800 pixels on each line, and 600 lines. You're really not interested in the number of lines, just the number of pixels on a line. In this case, the value of "P" is 800. If, using the ruler, you measured the screen width to be 12 inches, the "horizontal resolution" is 800 pixels divided by 12 inches, or about 67 pixels per inch. This is a handy thing to know when creating stereograms.

The viewability of a stereogram depends greatly on the number you use as your initial pattern width. The only thing that really matters here is the actual linear measure of the pattern width, in inches. This is important because you typically measure the distance between your two eyes in inches, rather than something obscure like pixels. In viewing a stereogram, the actual distance between your two eyes also plays a part. Because all humans share a similar distance between their two eyes, you can be fairly safe in assuming that it's a constant value.

What makes a stereogram viewable is the linear distance between repeating patterns, compared to the distance between your two eyes. If you assume that everyone's eyes are the same distance apart, the only thing that really matters is the distance between repeating patterns.

Most successful stereograms use an initial pattern width anywhere in the neighborhood of three-quarters of an inch to an inch and a half.

TIP: A good rule of thumb is to shoot for a starting pattern width of about an inch. From there, you can always adjust the pattern width parameter to make it larger or smaller. In the example given above, the user worked with a monitor that displayed about 67 pixels per inch. For this user, 67 is a good number to start with. This creates a starting pattern width of one inch.

Figure 5.1 is an example of a simple stereogram created with a pattern width that translates to about one inch on this printed page.

Because your stereogram software probably specifies the pattern width in pixels, it generally is up to you to do the translation. Using the preceding "one-inch rule of thumb," the translation is pretty easy. If, for example, you want to print your image at full resolution on a 300 dpi printer, 300 pixels is probably a good choice for a pattern width. You can always experiment by creating images using values larger and smaller than this.

The general rule for viewability is that the larger the pattern width, the more difficult it is for the image to be viewed divergently. The smaller the pattern width, the easier it is. You can add a real challenge to a stereogram by using a relatively large pattern width. A pattern width of, say, two inches, makes it *very* difficult to see. Of course, the ***absolute maximum*** pattern width for normal divergent stereograms is somewhat smaller than the distance between your two eyes.

Figure 5.2 contains the same object as figure 5.1. The only difference is that the pattern width is increased to a value that translates to about two inches. Can you see it? Many people can't!

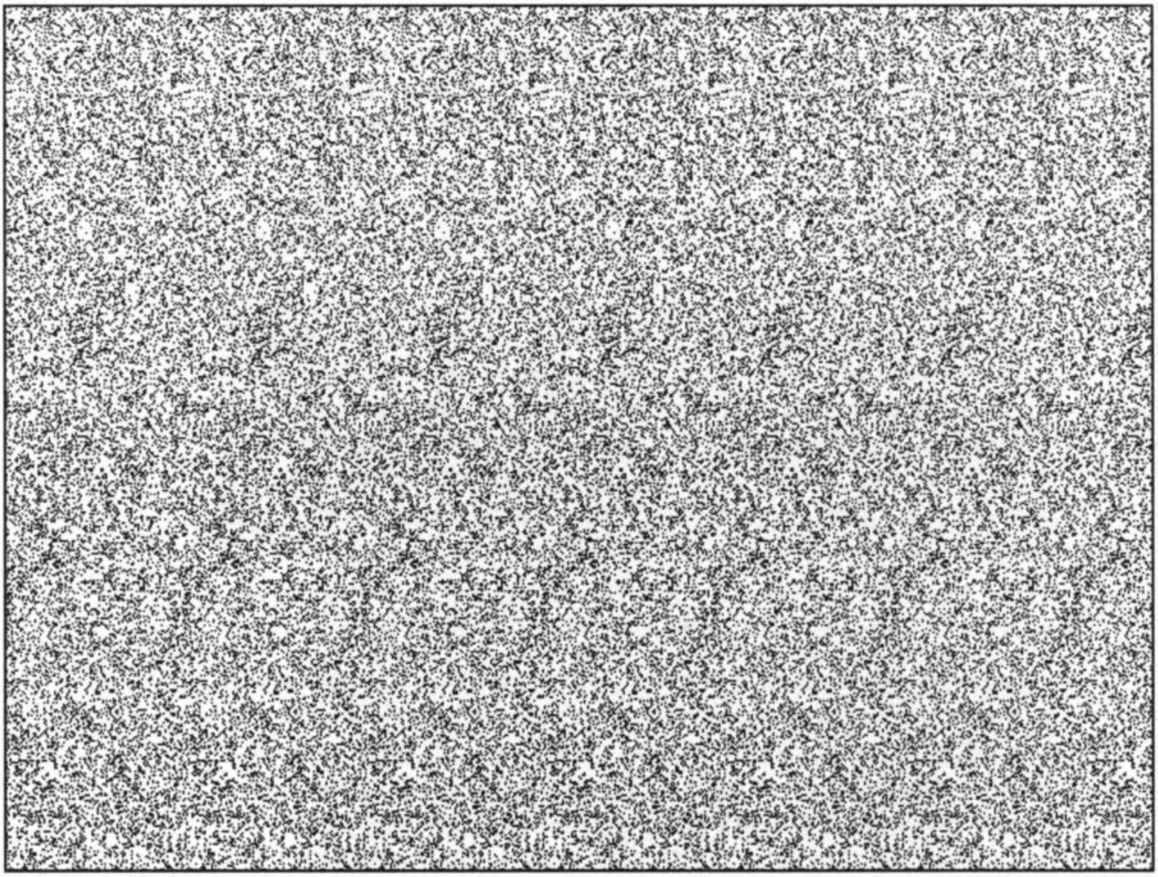


Figure 5.1

Stereogram with approximately a one-inch initial pattern.

TIP: If you're one of those folks who has difficulty viewing stereograms, start with a pattern width of a half-inch or so. After you master divergent viewing with this small pattern width, try larger values.

An interesting point in viewability is that if the pattern width is made too small, the image can become so "easy" to view that it actually becomes difficult. It's very difficult to "lock in" on a stereogram made with a very tiny pattern width. A good example of this is the POPOUT-LITE icons. These icons are actually very small stereograms.

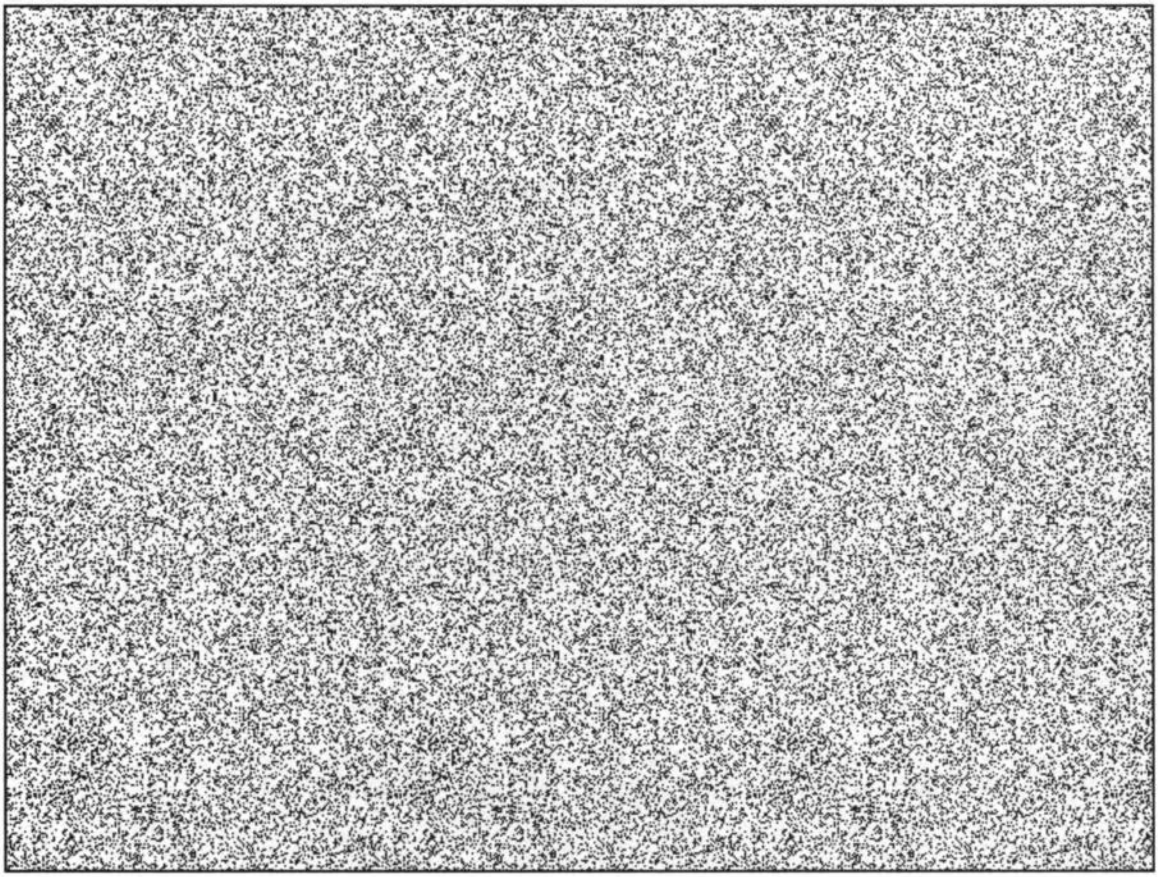


Figure 5.2

Stereogram with very large initial pattern.

Because these icons are only 32 pixels wide, we used a VERY small pattern width to create these miniature stereograms. The pattern width of these icons was chosen to be 14 pixels wide! Depending on the size of your monitor, they may be very small, and very difficult to see. By the way, POPOUT-LITE was used to create its own icons!

The key to divergent viewing is that while your left eye is looking at one object, or one point in the stereogram, your right is looking at an object (or point in the stereogram) one pattern width to the right. If the pattern width is too small, your right eye conceivably can be looking at a point that is two, three, or even more pattern widths to the right. The result is that mostly you see junk. You want your right eye to be looking only one pattern width away from your left eye.

Figure 5.3 was created with the same state of Texas object shown in figures 5.1 and 5.2, but this time a very small pattern width was used. Can you see it without ghost images or multiple levels? It's not easy!

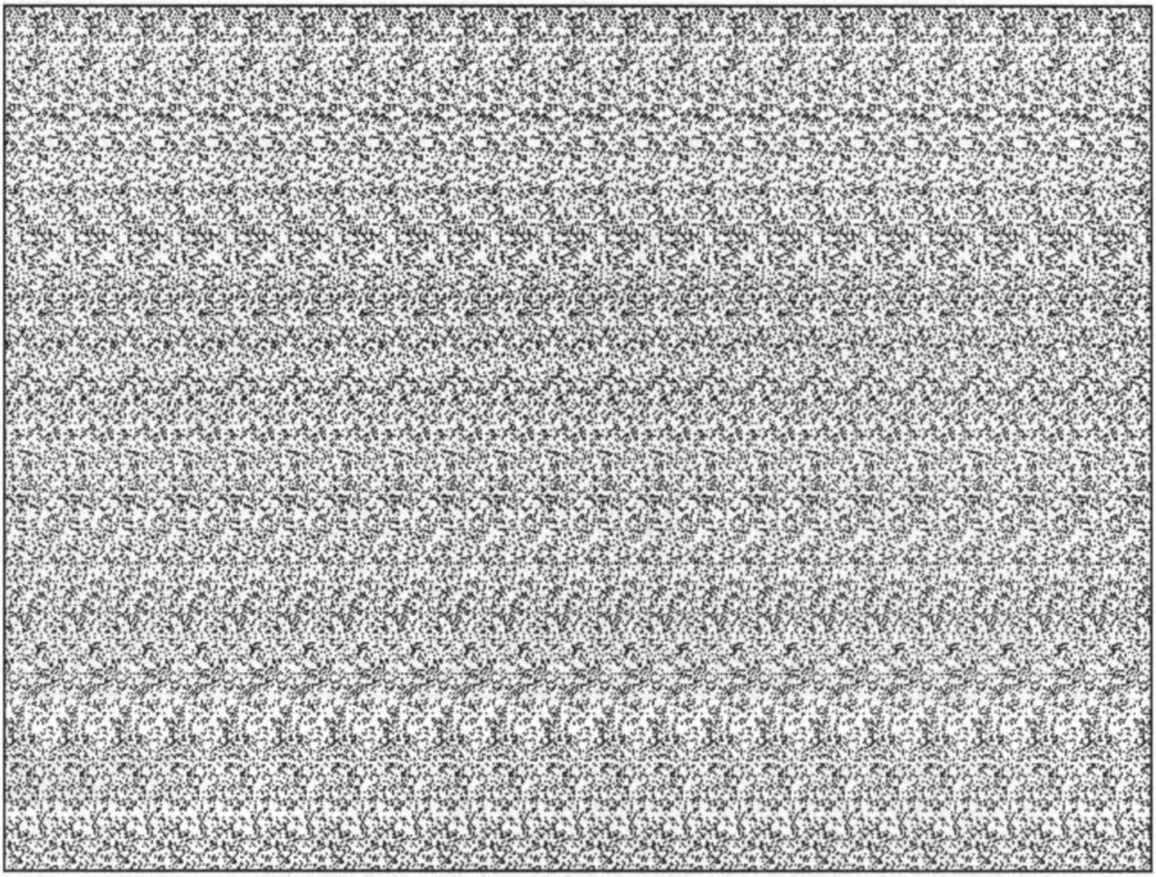


Figure 5.3

Stereogram with a very small initial pattern width.

TIP: If you create a stereogram that looks like it has double images or ghosts, or has more levels in it than you intended, try increasing the pattern width parameter and generate it again.

Aside from a stereogram's viewability, the pattern width parameter plays an important part in the total number of levels that can be contained within the image.

Recall from chapter 3 that a level of depth is created by either deleting or inserting an element into the pattern. Most software starts out with the initial pattern, and then deletes pattern elements to create levels of depth coming closer and closer to the viewer.

With some software, it is conceivable that you have more levels of depth in your depth file than you have pattern elements available to delete. Later, this chapter discusses a concept in which the stereogram software scales the intensity of the depth file. But for now, realize that you can't have more levels of depth than you have pattern elements.

You must get to know your software. Some software specifies the initial pattern width as the pattern width used to create the apparent points closest to you. In other software, the parameter specifies the pattern width used for the farthest points. With POPOUT-LITE, the pattern width that you specify is the pattern width that is used wherever there is black in the depth image. In other words, you're specifying the maximum pattern width that is used in the stereogram.

TIP: If your stereogram has many horizontal streaks that give you an outline of the objects in your depth image, it probably means that your pattern width is too small to handle this particular depth image, or your software doesn't have the capability to scale the intensity of the depth image. Unless you can tell the software to scale the depth image intensity, your only real recourse is to increase the pattern width. While POPOUT-PRO has this capability of scaling the intensity values of the depth image, POPOUT-LITE does not. If you see streaks, simply increase the pattern width, and then regenerate the image.

Go back and compare figures 5.1 through 5.3, and notice how many more levels there are as the size of the pattern increases.

The pattern width affects the amount of apparent depth in the image in yet another way. The value of the pattern width aids in determining how "deep" the image seems. For example, say that you start with a depth image that has 16 levels. Furthermore, assume that you're working with a piece of stereogram software that doesn't scale the intensity of the depth image. Sixteen levels really aren't many, so it doesn't matter much what your initial pattern width is. Now, generate two stereograms: one with a pattern width of 50, and another with a pattern width of 80. All 16 levels of depth are apparent in both images. The difference is that the image with an 80-pixel pattern width appears to have more depth than the 50-pixel image. It's just one of those things you may want to work with to get an image to your liking.

TIP: Even if you have a viewable image, experiment with the pattern width parameter, regenerating the image with various pattern widths near the one you chose. You may end up with an image you like even better.

Compare color plates 3 and 4. Both of these images were created using POPOUT-PRO, and are identical except for the pattern width. The image of color plate 3 has a pattern width that is much narrower than that of plate 4, and therefore is easier for many people to view. Notice also that color plate 4 seems to have more apparent depth than the image of plate 3, due to the wider pattern width.

Pixel Density

A very useful parameter in the generation of random dot stereograms is the *pixel density,* sometimes referred to as the *pixel ratio parameter.* As with many other parameters, this one comes in different disguises, depending on the stereogram software being used. Many stereogram generators don't even offer this as a parameter. The parameter only carries any real meaning in the creation of random dot stereograms (as opposed to colorfield images). POPOUT-LITE allows you to adjust the pixel density of your stereogram.

This parameter is presented in a generic sense, so you can understand the concept and are able to adapt this information to your own software. To keep things simple, POPOUT-LITE uses the same definitions of pixel density that are discussed here.

A simple black-and-white RDS has a total number of black pixels and a total number of white pixels. The sum of all the black pixels plus all the white pixels is the total number of pixels in the image. The pixel density parameter is defined as the percentage of white pixels in the image. In other words, the pixel density is the total number of white pixels in the image divided by the total number of all pixels in the image, multiplied by 100.

If you agree with this definition for now, you can see that if your software does not offer this parameter, what it really does is fix the pixel density value to 50 percent.

NOTE: Some stereogram software lets you specify the pixel density, but in a more indirect manner. For example, some software lets you specify that you want "three white pixels generated for every one black pixel." What they're really doing is letting you specify the pixel density by biasing the substitution pattern selection. For every four pixels created, three are white. Another way of expressing this is (3/4)*100 = 75 percent pixel density.

A pixel density of 50 percent is the simplest setting and just means that, statistically speaking, you're going to get about the same number of white pixels as you have black pixels.

As some of you may realize, this parameter affects the pattern substitution phase of creating the image. Now, focus on black-and-white RDS images. After the pattern for a line of the image is calculated, you must come up with a set of random black-and-white dots to substitute for the individual pattern elements.

The pixel density parameter is used to bias this selection in such a way that you get the desired percentage of white pixels with respect to black pixels. Choosing a substitution pattern is a random process, so this percentage may not be exact; it's really just a function of the software that chooses the random pattern for the substitution. If you refer back to Algorithm 4.7 in chapter 4, you see what I'm talking about. Assume that when you generate a random pixel, it's always either black or white. This algorithm shows how a random substitution pattern is biased by asking if the selection of each dot is a "good choice." If the software always answers, "Why, yes, it's a fine choice" to this question, then the result is a 50 percent pixel density. Otherwise, the answer to this question depends on how many black-and-white pixels are chosen so far, and what the user specified the pixel density to be.

One of the nicest uses for this kind of parameter is to print your stereogram. In the stereograms presented so far in this chapter, all generate with a pixel density set to 75 percent. This means that in all these images, three quarters of the pixels are white, and only one quarter is black.

The reason for this is to keep the images from being too dark. Because most people can't afford the high-quality color printers necessary for printing stereograms in full color, most stereograms created on a PC are printed in black and white. Changing the pixel density enables you to adjust how dark the image appears. This is especially true for very high-resolution prints.

> **TIP:** If your black-and-white printed RDS image looks too dark, try increasing the percentage of white pixels by increasing the pixel density and regenerate the image.

Aside from affecting the darkness of a hard copy, adjusting the pixel density can make for some very interesting images. Some interesting experiments can be done to test the limits of the human brain's ability to resolve a stereogram given only a very limited amount of information.

Translated into English, this means you should try experimenting with very large and very small pixel density values and see how few black pixels you can actually generate and still be able to view and resolve the image.

The image in figure 5.4 was generated using a pixel density of 98 percent, therefore only 2 percent of the dots are black. Can you still see it?

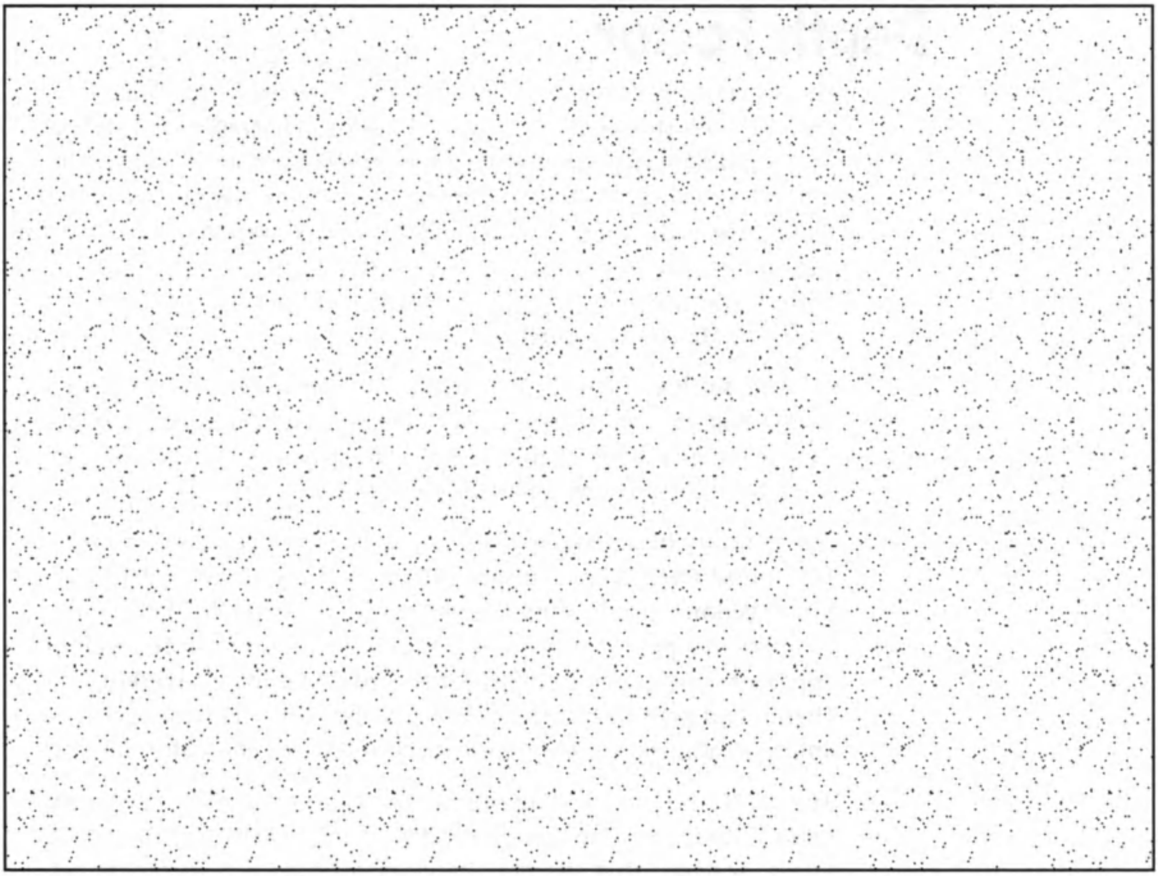


Figure 5.4

RDS with only two percent black dots.

The important thing here is that the pixel density is simply a mechanism to aid in biasing the selection of a random substitution pattern.

Although the effects of pixel density can be easily seen on black-and-white images, the same applies to color images. Again, you have to get to know your own software to understand how to tell it to get just what you want. In POPOUT-LITE, the pixel density is defined as the percentage of foreground pixels that appear in the stereogram. For black-and-white images, white pixels are considered to be the foreground pixels. For custom color RDS images, the foreground colors are specified by the user when creating the color pattern.

Depth Factor

Because the whole motivation behind stereogram generation is to create the illusion of depth, a most important ingredient in generating the image is a parameter that specifies the amount of apparent depth impressed into the final image. As you saw earlier, the pattern width affects the total apparent depth in the image. Actually, depending on how the stereogram software interprets the depth image, the pattern width may be the only mechanism available to you for adjusting this total apparent depth.

In some software, some sort of parameter, separate from the pattern width, enables you to do just this. Like all stereogram parameters, it may come in a number of disguises and under many different names. We refer to this parameter as the *depth factor.*

The majority of stereogram generators do not offer a depth factor. With most stereogram software, adjusting the amount of depth in the image is done by adjusting the pattern width. POPOUT-LITE is an example of a program where this is the case. With only 16 levels of apparent depth, such a parameter is considered to be "overkill." Higher-end programs such as POPOUT-PRO do offer the user the ability to adjust apparent depth without altering the pattern width.

The depth factor parameter only has any meaning when the stereogram software has the capability to scale the intensity of the depth image. Software that can't do this does not have any such parameter. With software that enables you to turn this scaling off or on, the depth factor only has any meaning when scaling is enabled.

For example, say you have a depth image that contains a grayscale image, and this particular image contains pixels with grayscale values between 0 and 255. (I chose these numbers because in many image formats, this is the case.) This means that your depth file has 256 total different pixel intensity values. This is really an ideal depth image, meaning that it contains the maximum number of possible grayscale values that many image formats support. It also means that you've given the stereogram software the maximum amount of information possible for creating a stereogram.

NOTE: Not all depth images contain pixels that span the whole grayscale range of 0 to 255. For images that contain only very dark pixels, you tend to want to use a large depth factor to create more depth. For depth images that contain only bright pixels, you want to use a smaller depth factor to keep from having too much depth.

If you go back to chapter 4 and review how the decision is made as to whether to insert or delete pattern elements, you may notice that there can be a problem with some stereogram software if it uses these algorithms as they are. For example, say that you create the depth image as described earlier, and now you want to create a stereogram with a pattern width of 100 pixels, suitable for your PC monitor. See the problem yet?

Remember, each time a pixel intensity increases as the stereogram software scans across a line of the depth image, an element must be deleted from the pattern. In other words, the value of the grayscale pixel is an indicator as to how many pattern elements need to be deleted. A depth image pixel with a grayscale value of 255 means that 255 elements must be deleted from the pattern, but the pattern is only 100 elements to begin with.

The result of something like this is a mess of ugly looking horizontal streaks in the image that sort of look like the outline of the objects in your depth image. The solution is to simply increase the pattern width to a value greater than the highest grayscale value in the depth image.

Actually, this also can be a problem. A big one, if you want to be able to see your stereogram on the computer monitor! With 256 grayscale levels, a pattern width of 300 or so, which is somewhat larger than the highest grayscale value, is recommended. Think for a second how wide a 300-pixel pattern width would be on a computer monitor. On some monitors, this is almost half the width of the screen! It's also quite a bit larger than the distance between your eyes, meaning that it's impossible to view the stereogram divergently.

Software that has a depth factor parameter of some sort enables you to avoid this problem. The idea is really simple, as long as the stereogram software has the capability to scale the depth file pixel grayscale intensity values. Scaling the depth file intensity values simply means that the software does not actually use the intensity of the grayscale pixel, but rather a scaled version of it.

Consider the earlier example in which 256 grayscale intensities were in the depth image, and you wanted to create a stereogram with a pattern width of 100 pixels. What if, when the stereogram software read the depth image, it simply divided the actual grayscale value by a constant such as four. Now, the stereogram software can effectively "fool itself" into thinking that instead of containing grayscale pixels between 0 and 255, the depth file really contains grayscale values between 0/4 = 0 and 255/4 = about 64!

At this point, the software can proceed normally and actually create the stereogram. Sixty-four elements can be deleted out of a 100-element pattern with no problem, and probably results in a very nice looking image.

TIP: When the stereogram software has the capability to scale the intensity of the individual pixels of the depth image before actually creating the stereogram, it allows for the possibility of generating a stereogram from any depth image, regardless of how much depth is conveyed in the depth image, and how wide the pattern is to be in the stereogram. It is in the user's best interest to create grayscale images that occupy the entire range of grayscale. The furthest points should have pixel values of zero, and the closest points should have pixel values of 255. This way, the stereogram software is being given the maximum possible amount of depth information and depth resolution to work with.

The idea of the depth factor is to tell the software just *how* to do this intensity scaling.

Instead of fussing with the division as you did earlier, you may find it more useful if the software hides all this from you. Personally, I don't much care about the details of what value it divided (or multiplied) each depth pixel intensity by. Just give me an image that looks decent, and a convenient way to control how much depth is there!

POPOUT-PRO is an example of software that scales the intensity of the depth image. The depth factor parameter is specified as a percentage. Specifically, the parameter defines the maximum percentage of the initial pattern width that is sacrificed for depth. In other words, it defines the maximum number of elements that can be deleted from the initial pattern width, stated as a percentage. A zero percent depth factor means that you are willing to sacrifice no part of your original pattern to create depth. In other words, you're telling the software not to modify the pattern. Of course, that's nonsense, because the result wouldn't have any depth at all. A 100 percent depth factor also is not useful. By telling the software that you're willing to sacrifice all the original pattern for depth, your "current pattern" can shrink all the way to zero!

For a more meaningful example, assume a depth factor of 64 percent. This means that when the depth image pixel intensity is a maximum white value of 255, the software scales it such that the resulting current pattern is 64 percent less than the original pattern width to create this highest level. With an initial pattern width of 100 elements, this means that 64 pixels are deleted. Notice that this turns out to be the same as the preceding example in which the depth file pixel intensities were divided by four.

The advantage of specifying depth factor as a percentage of the initial pattern is that you don't need to be concerned with the actual dividing or multiplying. If you specify a depth factor of, say, 80 percent with the preceding example, depth image pixels with a grayscale value of 255 result in the deletion of 80 percent of 100 = 80 elements from the pattern. Depth file pixels of other values are then scaled accordingly. This gives you a nice, easy mechanism to specify to the software just how much apparent depth you want in the image, and you don't need to worry about the actual pattern width that is used to create the image.

You must understand that with this philosophy, both the pattern width parameter as well as the depth factor parameter determine the total number of levels in the stereogram that you create. This may seem a bit confusing at first, but in practice it maximizes the options available for creating stereograms of any size and of any resolution.

In the preceding example with a pattern width of 100 pixels and a depth factor of 80 percent, a maximum of 80 levels of depth can be in the image. If you print the image to a 300 dpi printer, 100 pixels translate to one-third of an inch and probably is too small. In this case, choose a pattern width of about 300, which translate to one inch. If you use the same 80 percent depth factor, there are 240 (80 percent of 300) possible levels of depth.

TIP: As the pattern width increases, the total number of possible levels increases. As the depth factor increases for a given pattern width, the total number of possible levels also increases.

Figures 5.5 and 5.6 are both RDS images, using the same pattern width and depth image. The depth image contains grayscale pixels ranging in value from 0 to 255. Both images use a starting pattern width of 100 pixels.

In figure 5.5, the stereogram software was instructed to sacrifice up to ten percent of this pattern width for depth. This means that the brightest pixel in the depth image translates to a pattern width of ten percent of 100, or ten. The maximum number of levels in the stereogram is, therefore, ten.

Figure 5.6, on the other hand, was created using a depth factor of 20 percent. With that same depth image and pattern width used in figure 5.5, this image has a total of 20 percent of 100, or 20 levels.

Compare color plates 5 and 6. Both of these images were created using POPOUT-PRO, and both are identical except for the depth factor used. Both have the same initial pattern width, but plate 6 has a much higher depth factor. You can see that the image of plate 6 has far more depth than that of plate 5. In fact, plate 6 was created with such a large depth factor that the Teapot is abnormally distorted.

Depth Factor and Image Distortion

You must understand one final concept about the depth factor. In chapter 3, you learned about inserting and deleting elements into or from the current pattern. You can think of this as distorting the current pattern. You begin with an initial pattern at the left-hand edge of the image and then proceeded to distort this pattern as you move to the right-hand side.

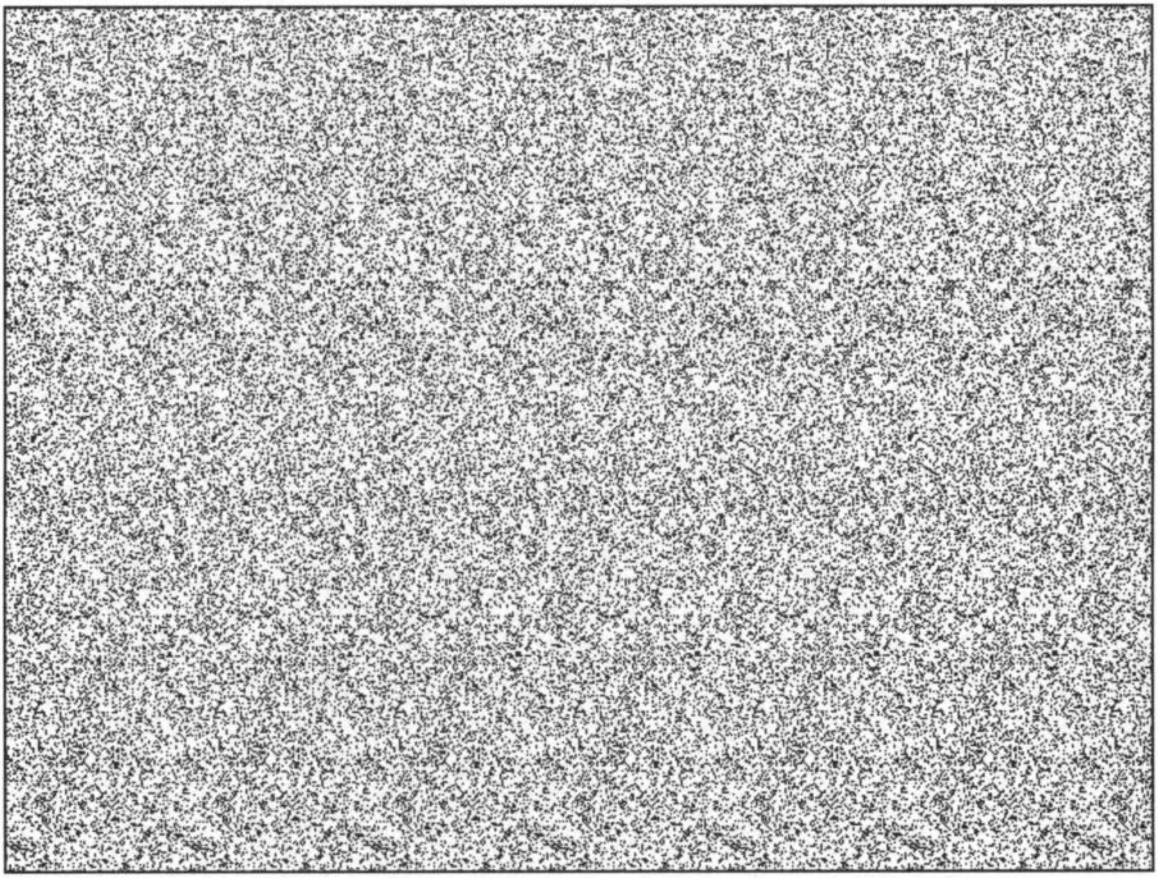


Figure 5.5

RDS that has a ten percent depth factor.

Depending on the actual contents of the depth image for that line, by the time you finish processing the line, the ending pattern may not even resemble the initial pattern. Think of this as a measure of the total distortion to the original pattern. The depth factor specifies the amount of this distortion.

The final step in creating a stereogram is to substitute the actual processed pattern elements with pixels. For random dot images, you simply substitute the pattern elements with a set of randomly selected pixels.

Because the pixels in this substitution set are random, it really doesn't matter how much the original pattern has been distorted. Who can tell a distorted set of random dots from a non-distorted set, anyway?

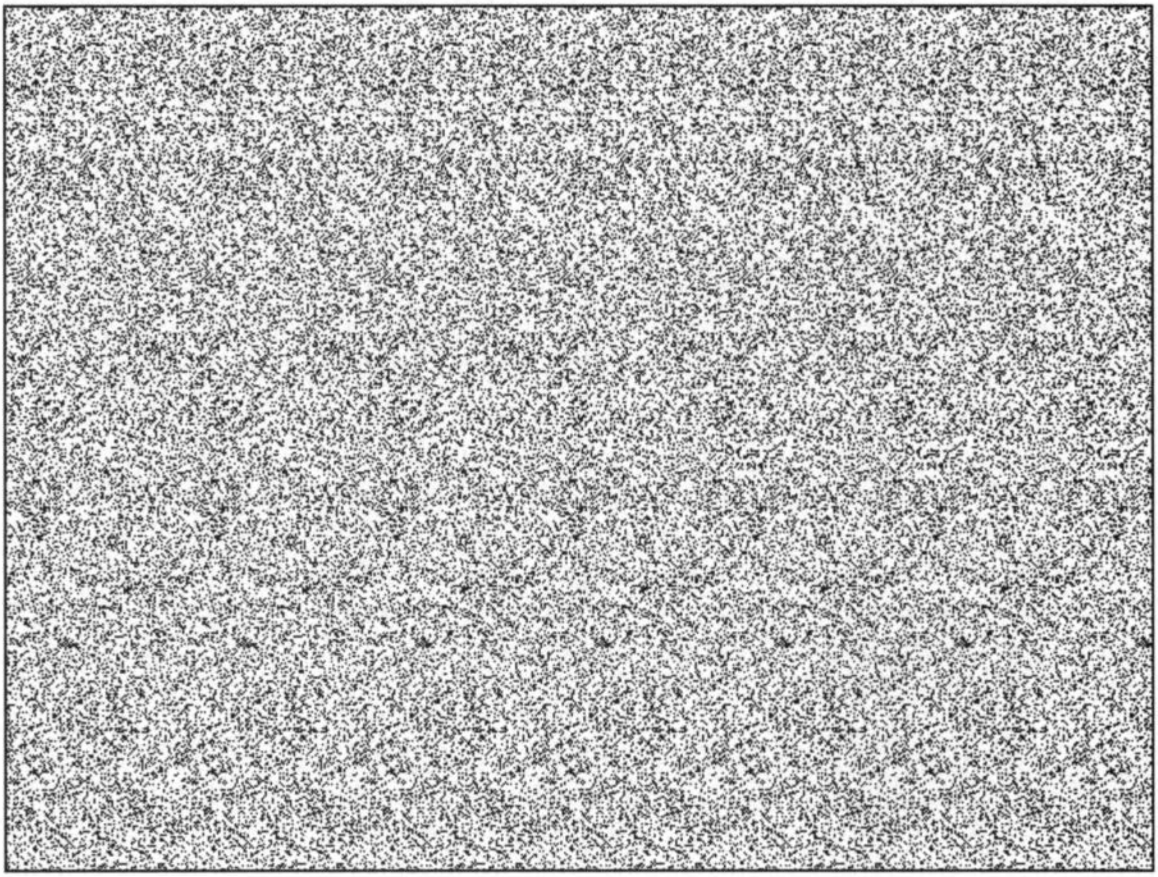


Figure 5.6

RDS that has a 20 percent depth factor.

The rules change a bit when generating a colorfield stereogram. In these images, the pixels used in the pattern substitution phase aren't randomly generated. Rather, they come from a second image—the colorfield image. If the image is processed from left to right, you notice in the stereogram that the tilings of the colorfield on the left aren't very distorted at all, but as you proceed to the right, the distortions may get greater and greater. This makes sense because the pattern is distorted as the image is processed.

The point of all this is that when generating RDS images, a great deal of distortion to the original pattern can be tolerated, seeing as how the actual generated pixels are random anyway. When generating colorfield stereograms, however, you really want to minimize distortion because it can become very annoying and degrade the quality of the image.

TIP: When generating RDS images, make the depth factor as high as you can get away with. As long as your hidden image looks good, the higher depth factor increases the total number of levels in the image. The random nature of the image tends to hide the distortion. Keep in mind, though, that more depth doesn't always mean a better image. For colorfield images, try to keep the depth factor as low as you can while still generating enough levels of depth to be able to accurately represent your hidden image. This may require increasing the pattern width.

Creating colorfield images can be a challenge. Because you want to minimize the total distortion of the colorfield, you want to keep the depth factor relatively low.

Fortunately, some stereogram software helps you with this. A number of packages offer the capability to begin processing the image in the center, rather than at one of the edges. This is very useful for creating colorfield stereograms. By beginning processing in the middle rather than at the edges, the total distortion is effectively cut in half; half the distortion appears at both edges, and the center of the image is the point of zero distortion.

TIP: If you're creating a colorfield stereogram, and your stereogram software allows for it, begin processing in the middle of the image rather than at one of the edges. This enables you to increase the depth factor, and therefore increase the total number of levels in the image. POPOUT-PRO offers the user the ability to decide whether the starting point for processing will be at the edge or in the middle of the image.

Another fortunate thing is in the way the human brain tends to be more forgiving about the total number of levels in the image when viewing a colorfield stereogram.

Chances are, the best stereograms you see are of the colorfield variety. Aside from being more interesting than an RDS, they just look better. When a colorfield stereogram is created using a well-chosen colorfield image, the eye tends to merge the edges of the individual levels so that it becomes difficult to actually identify the separate levels. Even when individual levels can be identified, your eye tends to concentrate on the colorfield pattern, and you don't notice the levels as much. This is especially true for high resolution color images generated with a very noisy colorfield.

This is nice to know because it usually means that you can use fewer levels in the image and still maintain the image quality. Given that you really want to minimize the depth factor with these types of images to minimize distortion, the levels of depth that you lose are less noticeable.

Compare figures 5.7 and 5.8; both contain the exact same hidden image. Figure 5.7 is a random dot stereogram, created with an initial pattern width of 100 elements. A total of 20 levels are in the image.

Compare this to figure 5.8. As you can see, figure 5.8 is a colorfield stereogram. This image uses a colorfield image with a width of 100 pixels, so that the pattern width is identical to that of figure 5.7. Notice how the individual levels of figure 5.8 are a little more difficult to identify. This is especially true when the colorfield image is in full color. With black and white, the effect isn't nearly as noticeable.

Processing Direction

Reference has already been made to the concept of processing direction, but this section covers it separately because it can sometimes improve image quality.

When the pattern for a line is created and processed, the ending pattern is a distorted version of the starting pattern for each line. Depending on the amount of apparent depth impressed into the line by distorting the pattern, the ending pattern may not even resemble the initial pattern. On lines where no change of depth occurs, the ending pattern is identical to the starting pattern.

This distortion of the original pattern means a distortion of the substitution set of pixels actually used to create the image. As explained earlier, this really doesn't matter for RDS images; you can't tell whether a random pattern is distorted. For colorfield images, it does matter because your eye definitely recognizes the distortion to the colorfields, especially if the colorfields are a recognizable image of some sort.

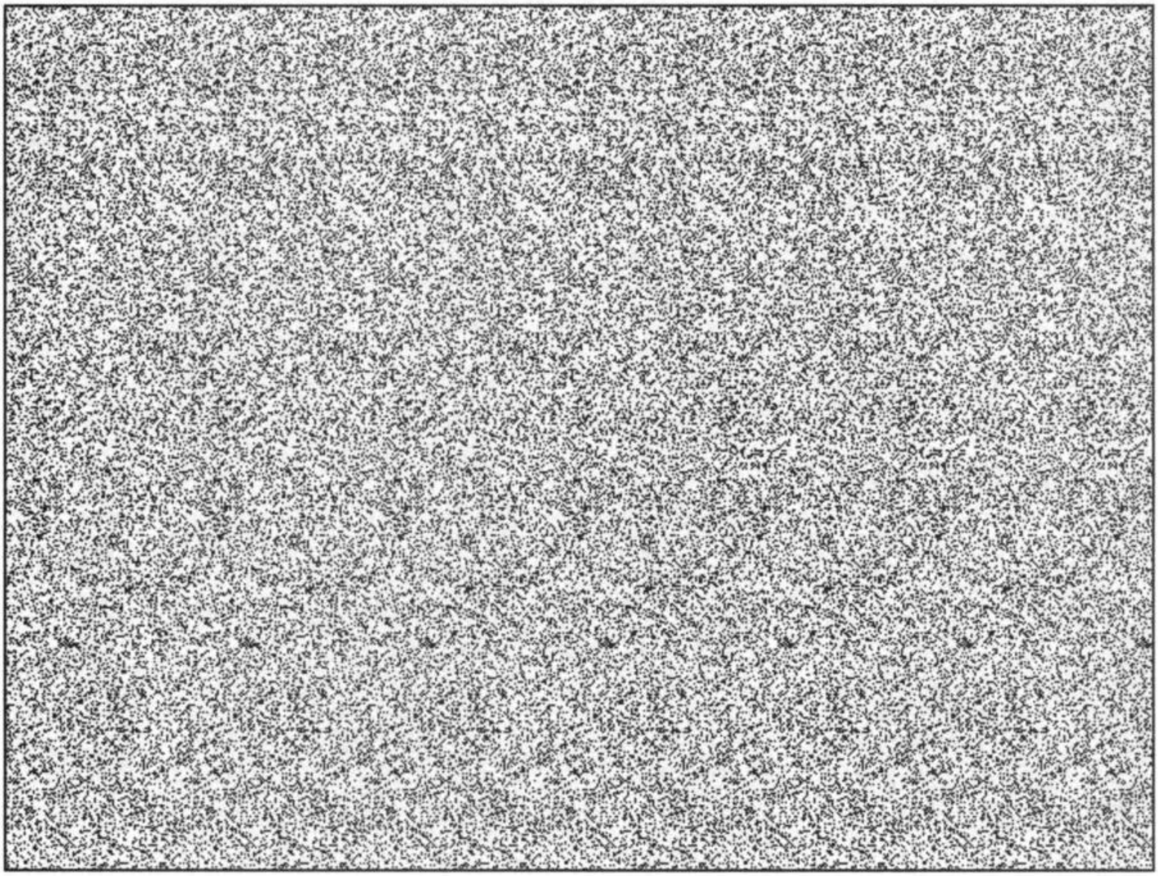


Figure 5.7

RDS image with 20 levels of depth.

Random Dot Stereograms are typically processed from one edge to the other—usually from left to right, although this really doesn't matter. The image could just as easily be created by processing the image from right to left.

This edge-to-edge processing is by far the easiest to implement in software and takes a bit less time to actually execute.

TIP: In general, RDS images should be, and usually are, generated by processing the image starting from one edge and working across to the other. POPOUT-LITE processes all images from left to right.

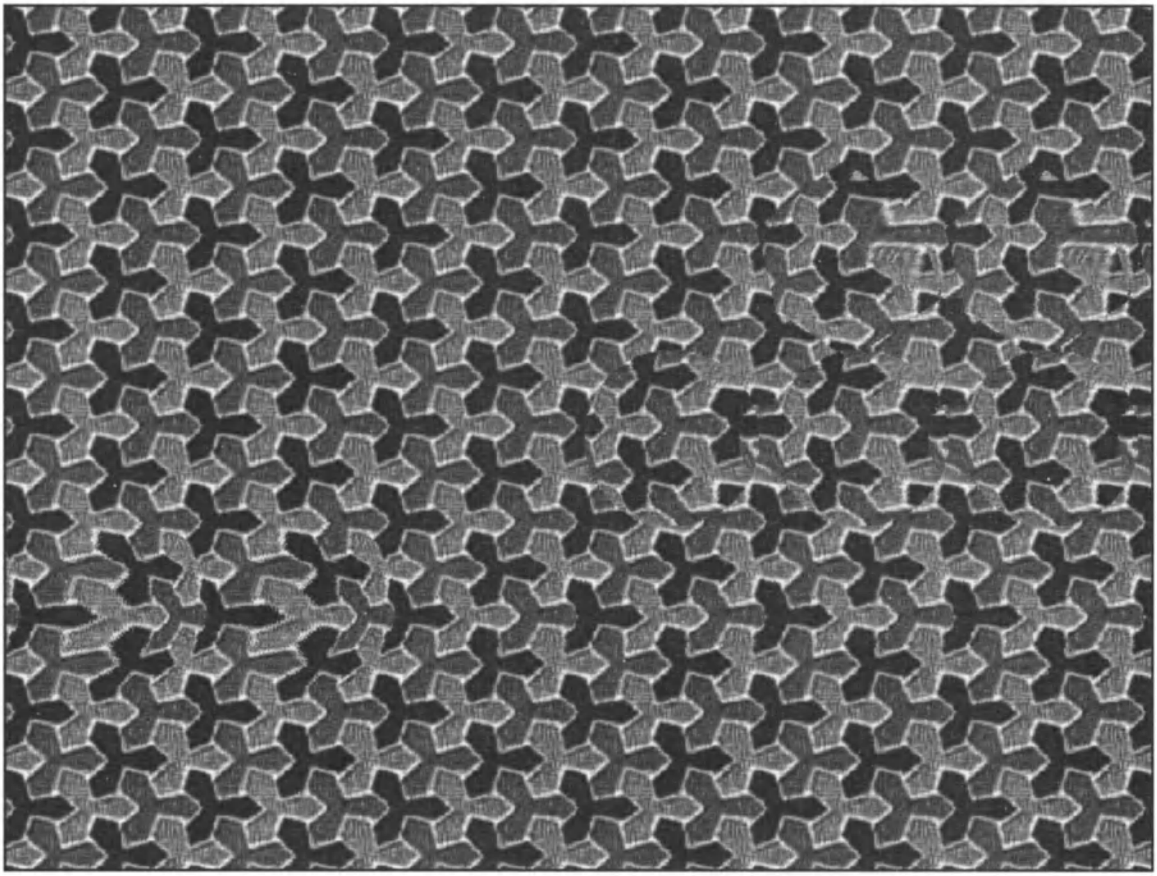


Figure 5.8

Colorfield stereogram with 20 levels of depth.

With colorfield images, you may want to go either way, depending on the specific colorfield image used to create the stereogram.

When using a colorfield stereogram generator such as POPOUT-PRO, just about any image can be considered as a candidate for use in a colorfield stereogram. Subsequently, all candidate colorfields for a stereogram fall into one of the following three categories:

- *The candidate colorfield is a tilable image.* This means that the image has been created in such a way that the right-hand side of the image "matches up" with the left-hand side. When multiple copies of a tilable image are set beside each other, no visible seam separates the individual instances of the image. A good example of tilable images are Window's wallpaper images. When a tilable image has been designed such that the top and bottom also match up, the result is usually a better looking stereogram than one where just the left and right edges match.

- *The candidate colorfield is a non-tilable image.* This covers just about everything else. These are just plain images, which can be either graphics or scans of photographs. If you place multiple copies of this kind of image side by side, a noticeable seam occurs between each copy of the images.
- *The candidate colorfield may be pretty, but it isn't really suitable for use as a stereogram colorfield.* Not all images work well with stereograms. A good stereogram colorfield image is noisy, meaning that the image has no large sections where all the pixels in that region are of the same color—sometimes called *flat fields*. Images with many of flat fields don't do well as stereogram colorfields. Some Window's wallpaper images make great wallpaper, but lousy colorfields, simply because of their flat fields. Before using an image like this, consider preprocessing the image using some sort of image processing program to add a bit of "noise" to eliminate large flat fields.

Of course, for generating stereograms, you're only interested in those images that fall into the first two categories.

Stereograms created using tilable colorfields look best when no visible seam is in the stereogram. For most stereogram software, the easiest way to accomplish this is to begin processing the image at one of the edges rather than in the middle. These images are almost always some sort of graphic, where more distortion can be tolerated. Many colorfield images are generated by using tilable colorfields and processing the image from edge to edge.

Stereograms using non-tilable colorfields tend to look best when processing begins in the middle and works out to both edges. Because many non-tilable images are of photograph scans, distortion is more noticeable than if it were a random-looking graphic of some kind. By starting processing in the center of the image, the point of zero distortion is right in the middle of the line. Because this is done for each line, the result is a line of zero distortion directly down the middle of the stereogram. Two lines of maximum distortion exist; one down the left edge, and the other down the right edge. Because only half the line is processed at a time, this total distortion on each edge is only half as much as it would be if the image were processed from edge to edge.

TIP: Colorfield stereograms created using a tilable colorfield tend to look better when they are created by processing the image from one edge to the other. Stereograms using non-tilable colorfields tend to look better when the image is processed from the center out to both edges.

Compare figures 5.9 and 5.10. Both of these images were created using POPOUT-PRO. In figure 5.9, a colorfield stereogram was created using a colorfield image that was meant to be tiled. Notice how no visible seams are in the image. The image was processed from the left edge to the right, so that the point of zero distortion is down the left edge, and the point of maximum distortion is down the right edge. Also, compare figures 5.8 and 5.9, which use the same colorfield image. If you look very closely at figure 5.8, you can see a vertical seam directly down the center of the image. In figure 5.8, this visible seam may not be considered as desirable as the seam down the center of figure 5.10, which uses a non-tilable colorfield image.

Figure 5.10 uses a colorfield image that is a scan of a photograph. The colorfield image is only 100 pixels wide and displayed in black and white, so it's a very low resolution image. Because the distortion would be much more noticeable in this type of image, especially because of the low resolution, I instructed the software to process the image starting from the center. Notice that you can see a distinct vertical seam directly down the center of the image. It's perfectly straight, indicating a point of no distortion in each line of the image.

Image Scaling

Many folks who generate stereograms on their PCs are very happy to view the results of their effort right there on the monitor. Unless you own a high-quality color printer of the dye sublimation variety, your monitor is probably the only means you have to view color stereograms that you create. Because of the way they dither colors, normal color ink jet or pen printers simply don't do a good job of printing stereograms.

Occasionally, however, you're bound to create an image that, even though it may be black and white, you'll want to print. Printing an image is usually associated with some sort of image scaling because the resolution of your printer, typically, is much greater than the resolution of your monitor.

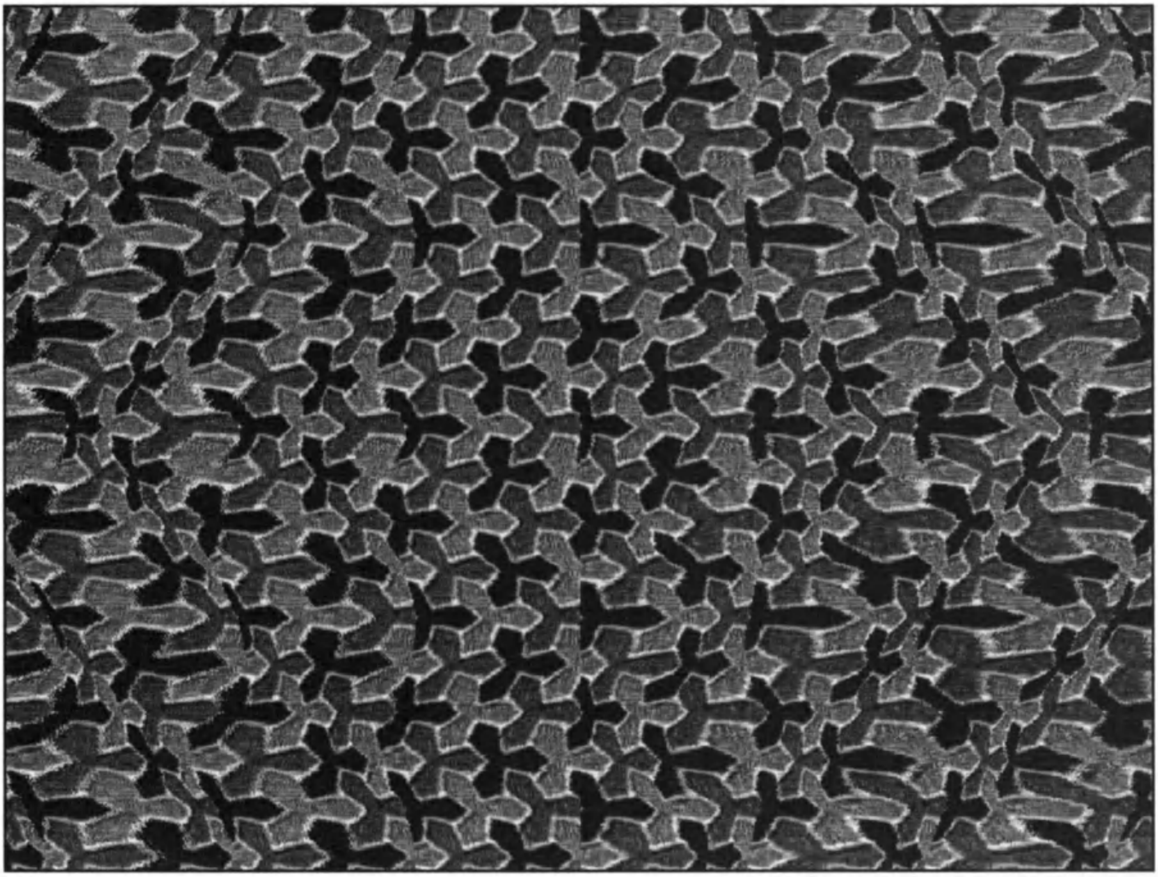


Figure 5.9

Stereogram using a tilable colorfield.

Basically, you can scale a stereogram in two ways:

- After generating a stereogram that is viewable on the monitor, you can use some sort of paint program or image viewer program to actually print the image. Most of these types of programs include a mechanism to resize the image for hard copy. Window's Paintbrush, for example, enables you to specify how an image should be rescaled before printing.

- The stereogram generating software can actually do the image resizing rather than the program used to print the image. This is done by scaling the depth image instead of scaling the output stereogram.

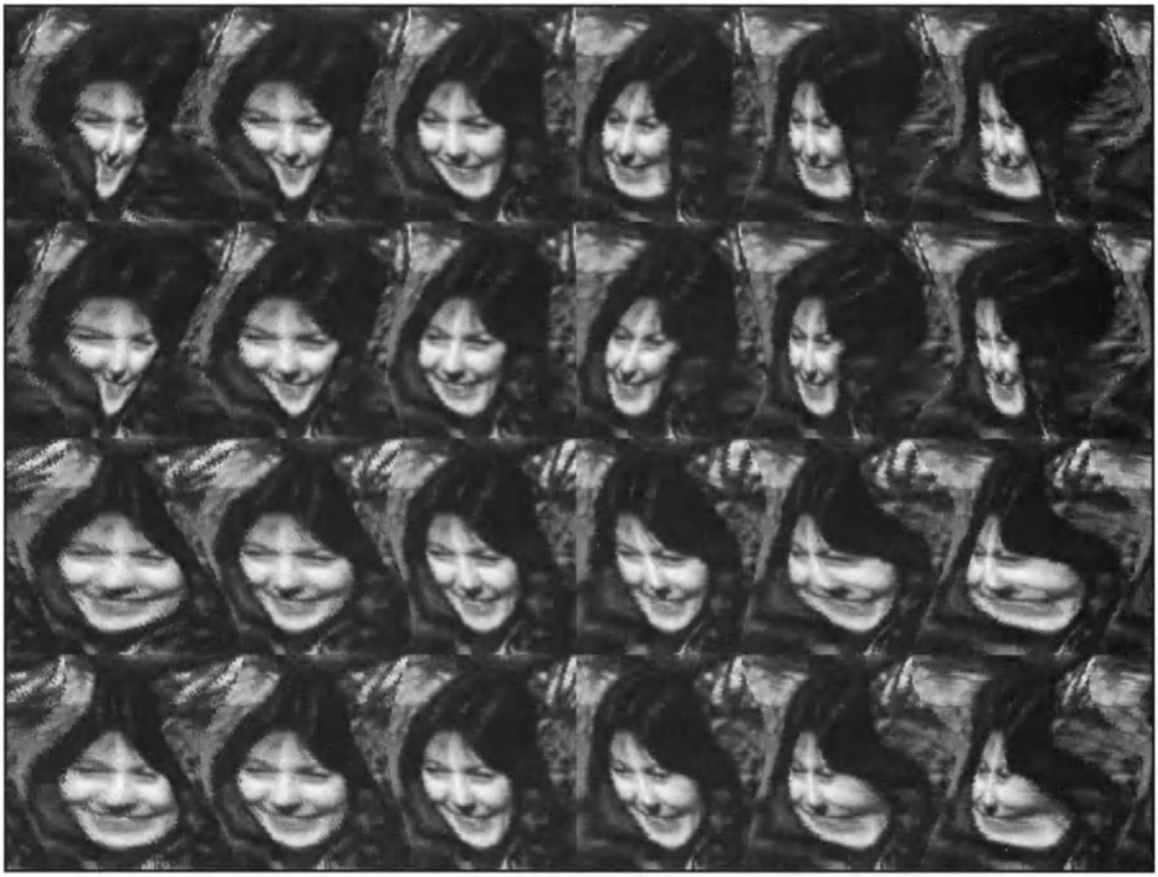


Figure 5.10

Stereogram using a non-tilable colorfield.

The first of these techniques is straightforward and has some definite advantages. Assume, for example, that you have a depth image 800 pixels wide by 600 pixels high. If your PC monitor is an 800 x 600 monitor, this image fills the screen and looks quite nice.

Now, say that you want to print your masterpiece to your 300 dpi printer. If you were to send the image to your printer without doing any scaling and the printer were to print one dot for each pixel, it would be printing 300 pixels per inch. In other words, your 800-pixel wide image comes out on paper at 800/300 = about 2.7 inches wide, and 600/300 = 2 inches high. Just a tiny little thing, hardly worth showing off!

It makes sense, then, to scale the image for printing. To create a hard copy that's a bit more viewable, say that you're shooting for an image that fills the space of a regular 8 1/2-by-11-inch sheet of paper. If you were to tell an image printer program (such as Paintbrush) to scale the image by 400 percent, the image would be printed at four times its original size.

Scaling an image by 400 percent (a factor of 4) is equivalent to telling the printer to print the image while pretending it has only a fourth of its real printing capability. In this example, it would be like printing to a printer with 300/4 = 75 dpi. At 75 dpi, an 800-pixel wide image would be 800/75 = 10.6 inches wide, and 600/75 = 8 inches high. On an 8 1/2-by-11-inch sheet of paper, this would look quite good.

Another way to look at this is that the print program duplicates the individual pixels before sending them to the printer. Rather than send the printer 800 pixels per line, it sends it 800 × 4 = 3,200 pixels per line, and 600 × 4 = 2,400 lines. On the same 300 dpi printer, this means a hard copy image of 3,200/300 = about 10.6 inches wide, and 2,400/300 = 8 inches high.

TIP: When scaling a stereogram for hard copy, always scale it by an integer amount. For example, don't scale a stereogram by 350 percent, or a factor of 3.5. If you do, this may result in unwanted vertical ridges, called *artifacts*, in the image. If you see these vertical ridges when viewing the image, it is often caused by improper scaling. If the program you're using to print the stereogram offers you the option to use the printer's resolution, use it! But, be sure to scale the image for printing as in the above example, where the image was scaled by 400 percent.

The very low resolution stereograms of chapter 3 were created in just this manner. POPOUT-LITE was used to generate a small, 80 pixel by 60 line stereogram. If you think about it, this is a pretty small image. But, using Paintbrush, the image was printed by scaling the image by a factor of about 30, or in other words, 3000 percent!

By scaling the image when it is printed, you sacrifice something. In the example above, you've got this great, 300 dpi printer, but are only printing at an effective 75 dots per inch, a fourth of that potential resolution. Because pixels are being duplicated, you're still only printing a stereogram that has 800 pixels on a line, when you could be printing a stereogram four times that size. Each pixel that appears on the hard copy could potentially be one fourth as wide and one fourth as high. This means that potentially, you could be printing an image that has four times as many pixels per line, and four times as many lines.

Remember, the more pixels per inch, the more possible levels of depth you can have in the stereogram. You could conceivably be creating stereograms with four times as many levels as what you just printed!

So now you might say that you can just create bigger depth images. Instead of creating 800 x 600 pixel depth images, you start creating 3,200 x 2,400 pixel images.

If you said this, you were right—technically. Of course, when you get down to actually doing it, you find that working with depth images that large can be very unwieldy, even impossible.

This brings you to the second way in which stereograms can be scaled. Some stereogram software realizes that not all your images are destined for the monitor and that you probably want to take full advantage of your printer's higher resolution to create the best quality hard copy of a stereogram possible on your printer.

This is done by scaling the depth image as it is being read by the stereogram software instead of scaling the image when printing. Because POPOUT-PRO is a more sophisticated stereogram generator that was designed for large, high resolution images, it has this ability.

As the stereogram software reads the depth image, it increases the image's size by a factor that you provide. For example, if you use the same 800 x 600 image mentioned earlier and direct the stereogram software to scale by a factor of four, then the stereogram software actually generates a stereogram 3,200 x 2,400 pixels! The nice thing is that you only had to create a depth image one fourth that size.

When it comes time to print an image of this size, you don't need to do any more scaling than the stereogram software has already done. That same 300 dpi printer generates a 10.6-by-6-inch stereogram. The difference is that 2,400 different pixels are on a line, rather than just 800 pixels magnified four times.

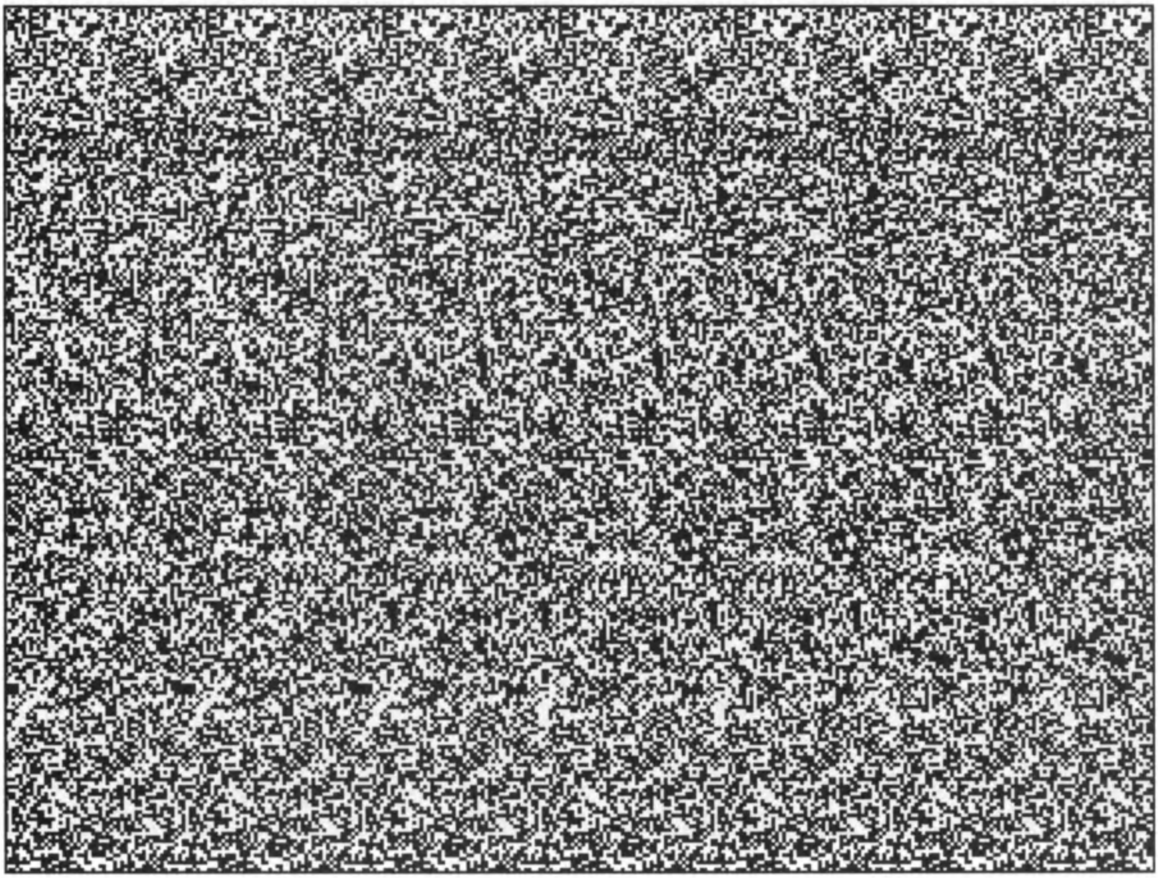


Figure 5.11

Stereogram resized after it was generated.

The advantage of this technique in stereogram creation is that it enables you to work with reasonably sized depth images, yet create large, high resolution stereograms. Because the stereogram software is scaling the depth image, this enables you to increase your pattern width by the same factor, resulting in many more levels of depth. Software that has this capability usually performs this scaling by pixel replication. If the stereogram software does a resampling, or interpolation on the depth image as it reads it, the resulting stereogram could potentially have many more levels of depth than were originally contained in the depth image. In the stereogram software, this might be referred to as filtering the depth image.

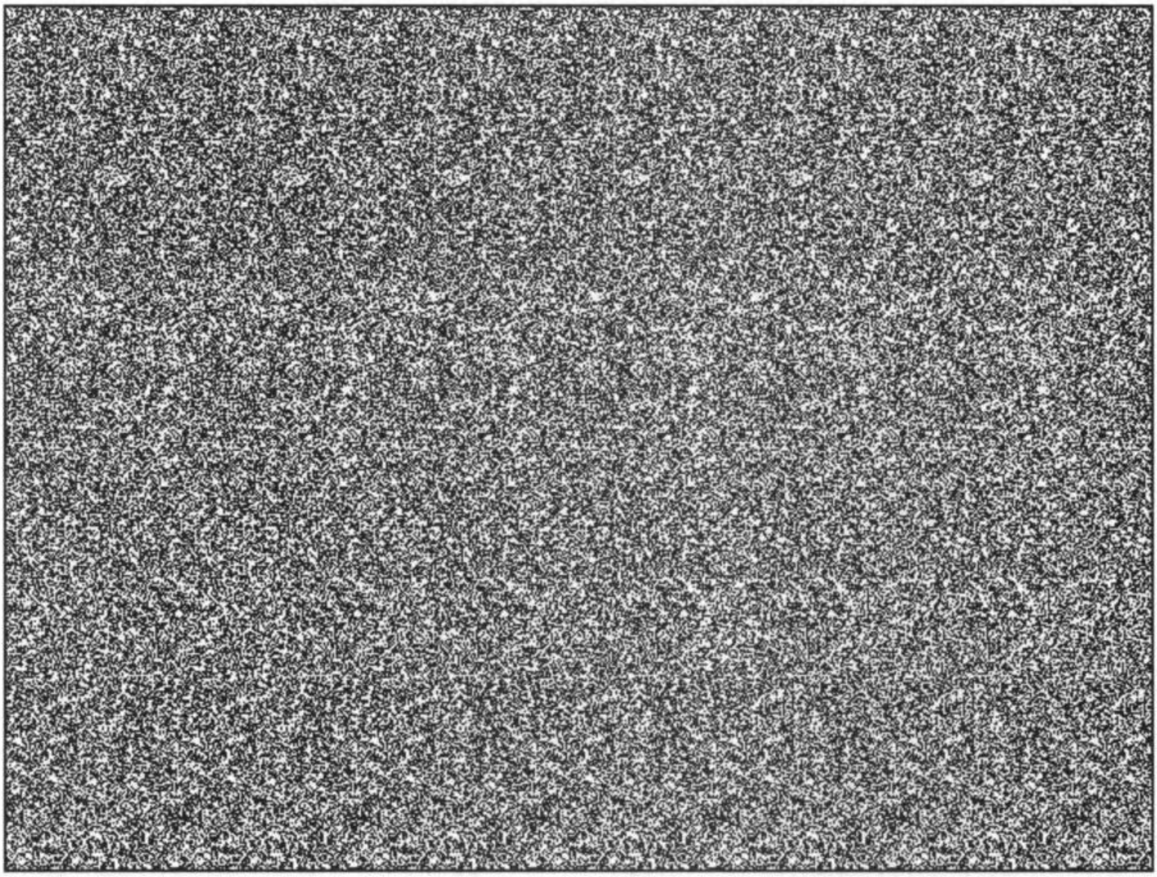


Figure 5.12

Stereogram created by scaling the depth image.

The disadvantage is that the stereogram files on disk are now quite a bit larger and chew up the empty space on your disk much faster.

TIP: Be really careful if your stereogram software has the capability to scale the depth image. The result can be a very large stereogram file, and the image can take a very long time to create!

To show you this concept, I've backed off a bit on resolution. Figures 5.11 and 5.12 are two stereograms that were created with POPOUT-PRO, using the depth image of 320 x 200 pixels. In figure 5.11, the image was scaled by two after the stereogram was created, resulting in 320 x 200 pixels that are twice as wide and twice as high as the originals. In other words, the individual pixels were magnified, so that the whole image is simply a magnified version of the original.

Figure 5.12, on the other hand, was generated by telling the stereogram software to double the size of the same 320 x 200 depth image before the stereogram was created. Although the resulting image is the same size (640 x 480) as figure 5.11, the resolution of the image is twice that of figure 5.11. Even though the pattern width of figure 5.12 is the same as that of figure 5.11 when measured in inches, it is actually twice the number of pixels used to create figure 5.11. The increased pattern width allows the stereogram software to generate more levels of depth.

Summary

This chapter presents common and useful parameters often found in stereogram programs. As mentioned earlier, there is really no standard for some of these terms, and some parameters fall under a number of different terms, depending on the software. How these parameters are offered to the user, or whether they're offered at all, is a function of the software and a choice of the software's author.

As mentioned earlier, generating a stereogram is a bit like turning on a television. All stereogram programs generate stereograms, just as all televisions generate images and sound. You just have to get used to your own software and find out which knobs are there for you to fiddle with. Sometimes, the best way to find out what a knob does is simply to turn it and look at the results. Just as different televisions have different sets of knobs, different stereogram software have different parameters.

The following list describes the parameters that this chapter addressed:

- Probably the most common of stereogram generation parameters, the pattern width is the key in creating a viewable or nonviewable image. While almost all stereogram software requires the user to specify this value in terms of pixels, what counts is the actual linear pattern width of the resulting stereogram. This linear pattern width depends on the resolution of the medium used to display or print the image, and should be somewhere between three quarters of an inch to an inch and a half.
- The pixel density parameter gives the user some control in how the random selection of substitution patterns will occur. Pixel density is a generic term that may take many forms, depending on the software used. By allowing the user to specify the statistical make-up of the color or intensity ratio of the substitution patterns, various types of random dot stereograms can be created.
- It is often desirable for the user to be able to control the total amount of apparent depth in a stereogram. This is done by using some sort of depth factor parameter, and gives the user precise control over the total apparent depth.
- By allowing the user to specify how much apparent depth to impress into the image, the stereogram software is in effect allowing the user to specify how much distortion to introduce into the original pattern for each line. While this has no noticeable effect on normal RDS images, it can greatly diminish the quality of colorfield stereograms. The effect of this distortion on colorfield images can be reduced by processing the image from the center, rather than from edge to edge.
- Many paint programs can be used to print stereograms that are stored in some standard file format. Typically, these programs have the ability to magnify, or scale, the image as it is printed. This technique is frequently used to produce larger, lower quality stereograms.
- Some stereogram software has the ability to perform this sort of scaling on the front end. That is, the depth image is scaled, or magnified, before the stereogram is created. While this method of scaling can produce some very large image files, it can also be used to create some very high quality stereograms.

6

Advanced Topics

In the previous chapters, I've tried to give you an introduction to the basics of stereogram generation. With what you've learned from these chapters, you now have the knowledge of how the stereogram illusion works, and how the computer goes about generating the image. This chapter discusses some of the more advanced and perhaps obscure concepts that are involved in creating the images.

Colorfield versus Random Dot Pattern Handling

You've seen how, within each line of a stereogram, raised regions are created by deleting an element or elements from the current pattern. The region is terminated by inserting new elements into the pattern.

In chapter 3, I emphasized the concept of inserting new elements, rather than reusing old, deleted ones. I included an algorithm in chapter 4 that does this very thing.

At this time, I'd like to introduce you to a second method of looking at the problem of pattern element insertion. This second method reuses old, deleted elements from the pattern to generate colorfield images. You may recall, a colorfield stereogram is one that has been created using a substitution pattern that does not consist of random dots. Rather, the substitution pattern is obtained from some line of a second, colorfield image. When the substitution pattern is a set of random dots, it really doesn't matter how many different elements you're left with in the final pattern. With a colorfield image, however, you're limited by the width of the colorfield. The total number of different elements in the final pattern can't exceed the width of the substitution line of the colorfield.

POPOUT-LITE is a RDS generator; it cannot generate colorfield stereograms. POPOUT-LITE inserts elements into the pattern by creating new elements instead of using deleted elements. POPOUT-PRO, on the other hand, is capable of creating both random dot as well as colorfield types of images, and must treat pattern insertion differently for these two types of images. Even though POPOUT-LITE does not create colorfield stereograms, we've included this topic to help you understand some of the important differences between processing the two different types of images.

By now you understand the concept of pattern substitution, the final step in creating the stereogram. If that substitution pattern is a set of random pixels, the result is some sort of random dot stereogram. If the substitution pattern is the data from one line of a colorfield image, the result is a colorfield stereogram.

Algorithm 6.1 is a "big picture" look at how colorfield stereograms are generated. In the algorithm, I've used a few variables which should be defined. CountY is a variable used to keep track of the line of the image that is currently being processed. ColorFldY is also a counter. It's function is to keep track of which line of the colorfield image will be used for the substitution pattern. ColorFldHeight is a variable that is passed to the algorithm that indicates how many lines are in the colorfield image. In many colorfield stereograms, the height of the colorfield image is less than the total number of lines in the stereogram. This variable, together with ColorFldY, will be used to determine when it's time to start over and re-read the colorfield image from the top.

Algorithm 6.1 Creating a Colorfield Stereogram

Step	Procedure
1	Set CountY equal to 0.
2	Set ColorFldY equal to 0.
3	Read a line of the depth image.
4	Process the image to create the final pattern.
5	Read line "ColorFldY" of the colorfield image.
6	Do a pattern substitution using the colorfield line.
7	Save the line of the stereogram to disk.
8	Increment ColorFldY.
9	If ColorFldY=ColorFldHeight, set ColorFldY to 0.
10	Increment CountY.
11	If CountY is less than HEIGHT, go to step 3.

Some of the steps and variables that are contained in algorithm 6.1 may seem a bit familiar. Step 1 initializes a variable called CountY to have a value of zero. Later, in step 10, CountY is incremented after a line is generated. In step 11, a comparison of CountY to the height of the stereogram is the determining factor as to whether the algorithm has finished processing the entire image.

Step 2 introduces a variable that I haven't used until now. This variable, called ColorFldY, is used as a counter. It's function is to keep track of which line of the colorfield image is being used in creating the stereogram. Notice that in step 8, the variable is incremented after a line of the stereogram has been generated. In step 9, ColorFldY is compared against the height of the colorfield image. If they're the same, it means that for the next line of the stereogram, the algorithm must go back and start at the top of the colorfield image, so ColorFldY is reset back to zero. This is what causes the colorfield to repeat in the vertical direction. Note that it is possible for the height of the colorfield image to be the same height as the stereogram. In this case, the colorfield image would never repeat vertically, as ColorFldY would never be reset back to zero.

NOTE: Some really interesting stereograms can be created by using a colorfield that is the same height as the stereogram. This technique is used by many stereogram artists. The colorfield then becomes a vertical strip within the image. Using a colorfield like this that is also tilable will usually help to create a very attractive looking image. By *tilable,* I'm referring to an image that is created for the purpose of being used as a tile. These types of images are created so that the right-hand side of the image matches up with the left side. The same thing occurs with the top and bottom. Color plates 3 and 4 are colorfield stereograms created with tilable colorfield images. Notice that there are no visible seams; you can't tell where one tile ends and the next one begins.

Step 6 of algorithm 6.1 does a pattern substitution, replacing all the elements in the final pattern with elements from one line (specified by ColorFldY) of the colorfield image. What does this imply with reference to the final pattern?

This implies that you've got just the right number of different element values in the processed pattern to exactly match the number of pixels in one line of the colorfield pattern! When inserting elements into the current pattern, the stereogram software must be sure not to use any more pattern elements than you've got colorfield width to later substitute. I mentioned earlier that the starting pattern width of a colorfield stereogram is defined by the width of the colorfield image. The goal is to have the total number of different elements used equal the colorfield width!

Algorithm 6.2 is a modified version of the pattern insertion algorithm introduced in chapter 4. I'm going to use the same variable names that I used in algorithm 4.6. CurPat is the computer's storage area for the current pattern. TempPat is a temporary work area. CurElemPtr is the index, or pointer, to the element of CurPat that is next in line to be used. NUMINS is a value passed to the algorithm that specifies how many elements need to be inserted. CurPatWidth is the width of the current pattern in CurPat. NewWidth will be the new pattern width after the insert has taken place. Finally, NewPtr is the pointer that is be used when redefining the pattern.

Algorithm 6.2 Pattern Insertion without New Elements

Step	Procedure
1	Copy all the elements in CurPat to TempPat.
2	Set CurElemPtr equal to(CurElemPtr – NUMINS), modulus CurPatWidth
3	Set NewPtr to 0.
4	Set Count to 0.
5	Set NewWidth to CurPatWidth+NUMINS.
6	Copy TempPat[CurElemPtr] to CurPat[NewPtr].
7	Increment NewPtr.
8	Increment CurElemPtr.
9	If CurElemPtr equals CurPatWidth, set CurElemPtr to 0.
10	If NewPtr is less than NewWidth, go to step 6.
11	Set CurPatWidth to NewWidth.
12	Set PatElemPtr to 0.
13	Done with insert!

Step 1 starts out just like the other algorithm, copying all the current pattern elements to a temporary area.

Things get a little different with step 2. Rather than create new elements, we use elements already in the current pattern. The effect of step 2 is to subtract the number of pattern insertions from the current element pointer, but to do it in such a way so it never creates a number less than zero. Instead, it wraps back around to the highest pattern element. The place where it stops is now the first element of the new pattern!

The remainder of the algorithm is identical to the steps in algorithm 4.6. The temporary area is copied to the current pattern storage, starting at the point determined by the modulus subtraction in step 2.

The result is that elements are inserted into the pattern, but those elements are made up of current pattern elements rather than new ones. Now, when you process a line, you can insert and insert, and still never introduce any elements into the current pattern that aren't there already!

Some stereogram programs use a slight variation on this concept—one which tends to lead to better results. Consider if I were to replace step 1 and 2 in algorithm 6.2 with those in algorithm 6.3.

Algorithm 6.3 Inserting Pattern Elements to Reduce Distortion

Step	Procedure
1	Copy all the elements in CurPat to TempPat.
2a	Set CurElem equal to CurPat[CurElemPtr].
2b	Set NewIndex equal to (CurElem – NUMINS), modulus ColorFldWIDTH.
2c	Set NewPtr equal to 0.
2d	If NewPtr = NUMINS, go to step 3.
2e	Copy the value NewIndex to CurPat[NewPtr].
2f	Increment NewPtr.
2g	Increment NewIndex.

Step	Procedure
2h	If NewIndex = ColorFldWIDTH, set NewIndex to 0.
2i	Go to step 2d.
3	Continue as in algorithm 6.2

Algorithm 6.3 looks a little overwhelming at first, but it really isn't all that bad. The following explains the basics of the algorithm.

In step 2 of algorithm 6.3, new elements are inserted by reusing the current pattern. Steps 2a through 2h of algorithm 6.3 also perform the function of inserting new elements, but do so in such a way as to maintain the original order of the pattern elements. Doing this tends to reduce the amount of distortion added to the image, and usually leads to better results.

This is a lot of information, and I want you to understand it. Perhaps the easiest way to compare algorithm 4.6 with algorithms 6.2 and 6.3 is to use a simple example.

Let's say that the original pattern was the ten elements `0123456789`. Let's also assume that we've done a pattern element deletion previous to this, and the current pattern is now `567891234`. As you can see, the current pattern width is nine elements, as I've previously deleted the zero.

Assume, furthermore, that the current element pointer is pointing to the one when we decide it's time to insert an element into the pattern.

As I showed you in chapter 3, the result of introducing a never-before-used element and redefining the pattern after the insert is the new pattern `A123456789`. I insert the `A`, and then copy the remainder of the pattern.

Using algorithm 6.2, the result of this insertion is `9123456789`. The first nine is the element that was inserted, and it came from the end of the current pattern. Because the current element pointer was pointing to the one, moving left by one place put us at the nine to begin the redefined pattern.

Using algorithm 6.3, the result is `0123456789`! Rather than use an element from the current pattern for the insert like 6.2, and instead of creating a new element like 4.6, algorithm 6.3 simply subtracts one (for one insertion) from the element pointed to by the current element pointer. That element was a one, so the inserted element was a zero. As you can see, this algorithm tends to reduce the distortion in the stereogram by being a little smarter in deciding just what the inserted elements should be. This algorithm tries to return the current pattern to as close to the original pattern sequence as possible. POPOUT-PRO is an example of a stereogram program that uses algorithm 6.3 to do pattern element insertions.

Other algorithms are used for pattern insertion, but most of these are beyond the scope of this book. Hopefully, the two algorithms I've presented here, along with the algorithms of chapter 4, will give you a better understanding as to some of the techniques that are used regarding pattern processing.

Convergent versus Divergent Images

This section is a brief introduction to the concept of the convergent stereogram. Whether you know it or not, you've already got enough information to create them! "So what is a convergent stereogram?", you ask. Good question. The term *convergent,* as well as the term *divergent,* are words that better describe a method of viewing, rather than a type of image.

Divergent viewing is what you've been doing when you look at the most common types of stereograms. I first introduced you to this in chapter 2. The term refers to the diverging of your eyes' imaginary lines of sight. As these imaginary lines diverge, your point of focus moves farther and farther away from you. When you view a stereogram divergently, your point of focus is somewhere behind the plane of the image. How *far* behind the image is determined by the current pattern width at any point on the image. The more divergent the viewing, the more relaxed your eyes.

Convergent viewing refers to the converging of these lines of sight. As the lines of sight converge, the point where they intersect gets closer and closer to you. When you view an image convergently, it means that this point of intersection (your point of focus) is somewhere *in front of* the plane of the image!

Figure 6.1 shows this. Compare figure 2.3 with figure 6.1. See the difference? In figure 6.1, the observer is staring at the row of objects in such a way as to force his or her point of focus in front of the objects, rather than behind them as in figure 2.3.

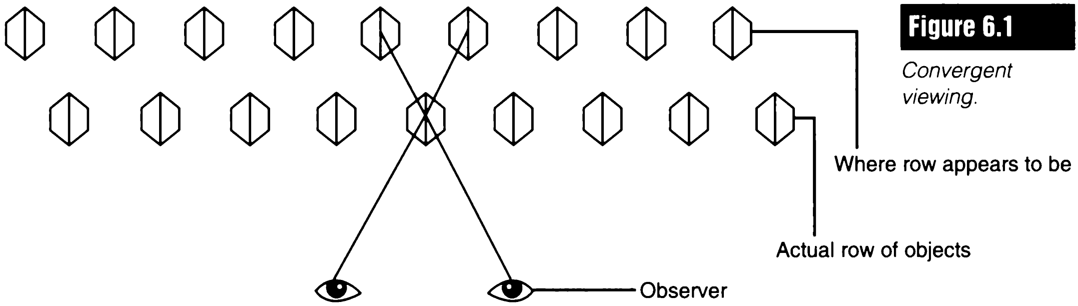


Figure 6.1

Convergent viewing.

Now that you've mastered divergent viewing, I'm going to throw in the proverbial monkey wrench and really mess things up for you!

Actually, convergent viewing is something you're very familiar with. Convergent viewing is achieved simply by crossing your eyes. Crossing your eyes simply means that your point of focus is very close to your eyes. Fortunately, viewing a stereogram in a convergent manner usually doesn't require your point of focus to be that close to you. As you may have guessed, viewing a stereogram in a convergent manner is more stressful to your eyes.

I'm going to show you a simple stereogram in figure 6.2. This is a picture of a sphere embedded in a wall. While not a really impressive stereogram, it works nicely for this example. Once you've viewed it in the normal divergent manner, try to view it convergently. This may take a little practice, but give it a try. What do you see?

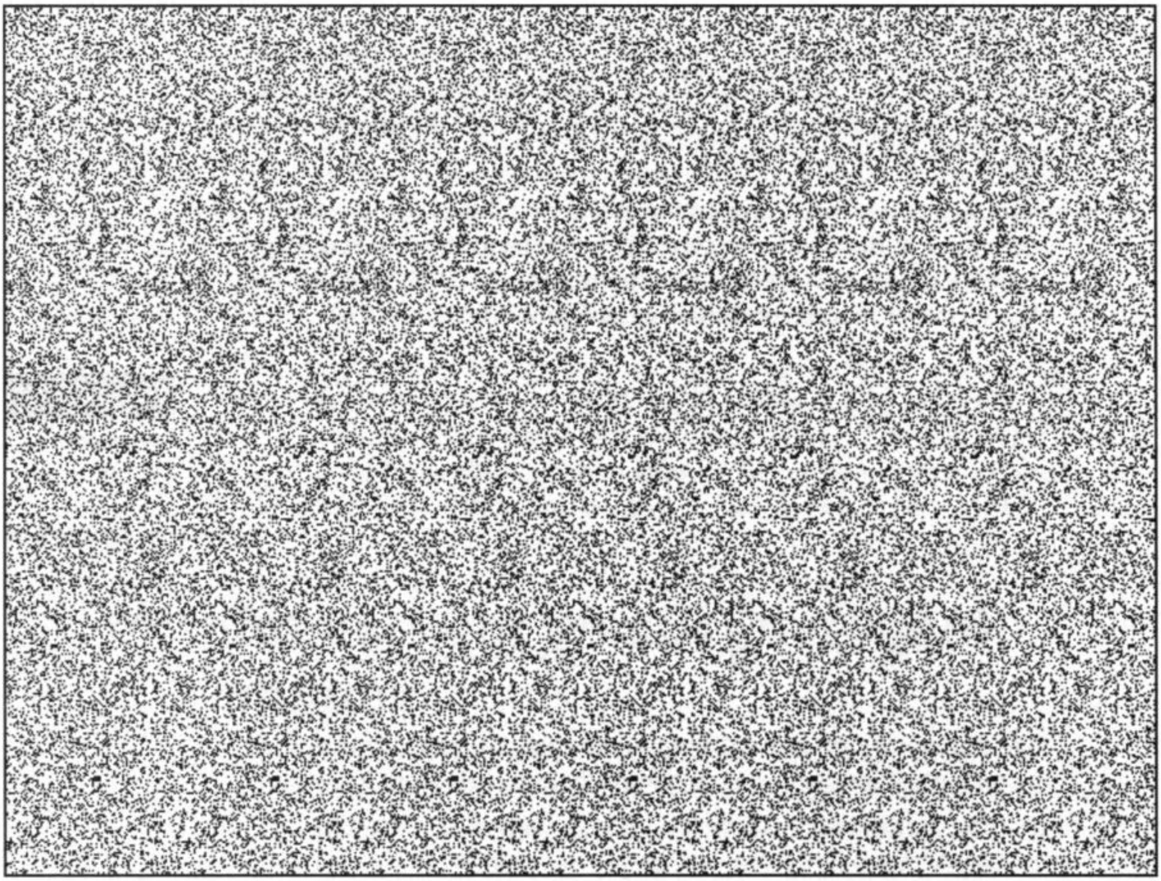


Figure 6.2

Simple stereo-gram of a sphere.

TIP: Try putting your index finger about halfway between your eyes and the stereogram. Stare at your finger so that you only see one finger. You are now viewing the stereogram convergently. Slowly move your finger toward you and away from you until you find the point where your lines of sight intersect the image a pattern width apart. You'll know you've got it when the image suddenly snaps into focus.

How does the image differ from when you viewed it in the normal divergent manner? If you've done it correctly, the result should be that when viewing the image convergently, you see what looks like a "mold" for the sphere. Instead of the sphere popping out of the wall, it's pushed into the wall!

In general, viewing a normal (divergent) stereogram in a convergent manner results in seeing all the levels of depth reversed. What used to be the closest points in the image are now the farthest from you. Try it with a stereogram you're already familiar with. When you view it convergently, you should see a mold of the image that you're used to seeing.

TIP: If we were sticklers about terminology, we might refer to common stereograms as "stereograms intended for divergent viewing" rather than "divergent stereograms." Consequently, the subject here is really, "stereograms intended for convergent viewing." Make sense?

Now, the next logical step is to create a normal stereogram not of a sphere stuck in a wall, but one where a sphere has been gouged out of a wall. I've done just this in figure 6.3. View figure 6.3 in the normal divergent manner. Can you see the mold of the sphere?

Now, try viewing figure 6.3 convergently. See it? Not only should you see a sphere stuck in a wall, but you should also notice something else. How is viewing figure 6.3 convergently different than viewing figure 6.2 divergently? In both cases, you see a sphere stuck in a wall.

The answer is that when you view figure 6.3 convergently, your point of focus is in front of the plane of the image. The result is the image looks like it's sitting there in space, somewhere between you and the page! Look at figure 6.3 convergently again, and see if you notice the effect.

Stereograms that have been designed to be viewed convergently, like figure 6.3, are generally more impressive. They're more difficult to see, as they cause some strain on the eyes, but the results are often worth the effort. Because your point of focus is in front of the image, the objects in the image appear to float in space as if you can reach out and touch them!

Realizing that figure 6.3 can also be seen as a gouge in a wall, some of you may have already figured out that creating a stereogram for convergent viewing is really quite easy. In short, all you need to do is create a normal stereogram of the mold of your object. When viewed convergently, the viewer will see the object.

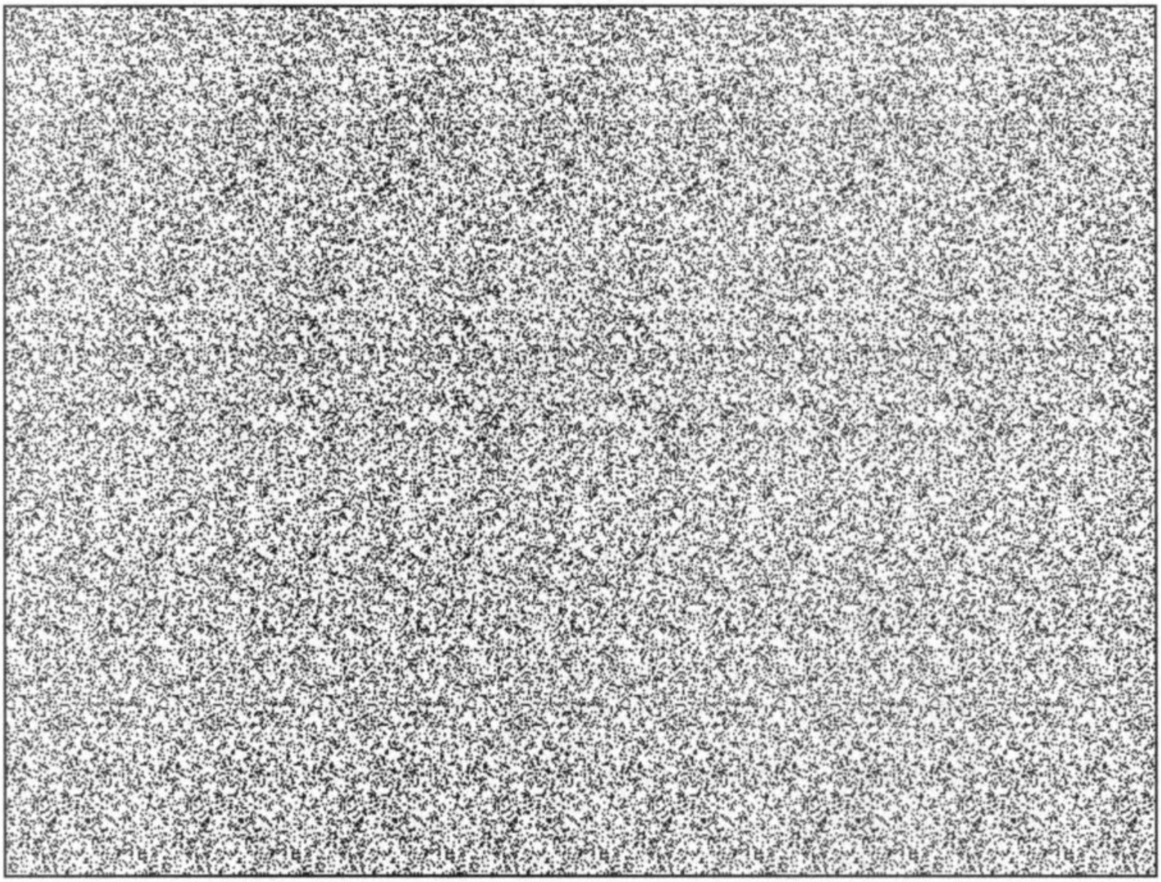


Figure 6.3

Simple stereogram of a mold of a sphere.

In short, you can use any stereogram software to generate this type of image. All you need to do is invert the intensities of your depth image—just reverse all the grayscales. I created figure 6.3 by taking the depth image that I used to create figure 6.2, and then reversed all the grayscale intensities. Instead of a white ball sitting in a dark fog, my depth image for figure 6.3 looked like a black ball sitting in a white fog. Some stereogram software automatically does this for you so that you don't have to go through this extra step of fussing with the depth image. POPOUT-PRO is an example of stereogram software that gives you the option to create either type of image.

This doesn't mean that you can't create convergent images with POPOUT-LITE! When using Paintbrush to create your depth image for POPOUT-LITE, you normally start with a Black canvas, then use the remaining 15 colors from the Paintbrush palette to start drawing the image. Dark Red produces an apparent level just above the black level, followed by Dark Green, and so on. The level closest to the observer is created by using White.

To create a convergent image, you simply need to reverse the order of the colors. Start Paintbrush with a White canvas, instead of Black. Then create your image by using the colors in the opposite order. For example, just after White comes Bright Cyan, followed by Bright Magenta, and so on. The resulting stereogram that you create with POPOUT-LITE will be a convergent image!

A very interesting aspect to this type of stereogram you may have not considered is that in a normal stereogram, there is an absolute limit on the width between repeating patterns. This distance must be somewhat less than the distance between the observer's two eyes.

If you go back and look at figure 6.1, imagine the objects being spaced a bit further apart, and then moved further from the observer. Because the point of focus is in front of the image, the objects can be wider apart than the distance between the observer's eyes! This pattern width limitation doesn't exist for this type of stereogram, which leads to a whole new realm of ideas in displaying the images.

TIP: Make a copy of a convergent stereogram on a piece of transparency plastic instead of paper. Put the transparency on an overhead projector, and project it on a wall or screen. A stereogram like this can be viewed convergently from quite a distance. This can make for a very interesting presentation if you give your audience a crash course in convergent viewing.

The bottom line of all of this is that there really isn't anything special about a convergent, or cross-eyed stereogram. The image was simply created using a depth image with all the intensities reversed. In the case of POPOUT-LITE, the Paintbrush colors are just used in the reverse order. The result was a mold of the object or objects when viewed in the normal divergent manner. Viewed convergently, the levels are reversed to yield the desired object.

Stereograms and Filters

In stereogram creation, it is often the desire of the artist to squeeze as much depth into the image as possible, without going so far as to distort the objects in the image from their intended shapes.

A number of techniques are used to increase the amount of apparent depth in an image. Two of the most common techniques involve *filtering*. For purposes of this discussion, I simply want you to think of filtering as smoothing or blurring an image. In the first technique, the depth image is smoothed or blurred before being read by the stereogram generation software. In the second technique, the stereogram is created normally, and then blurred as a final step.

Filtering the Depth Image

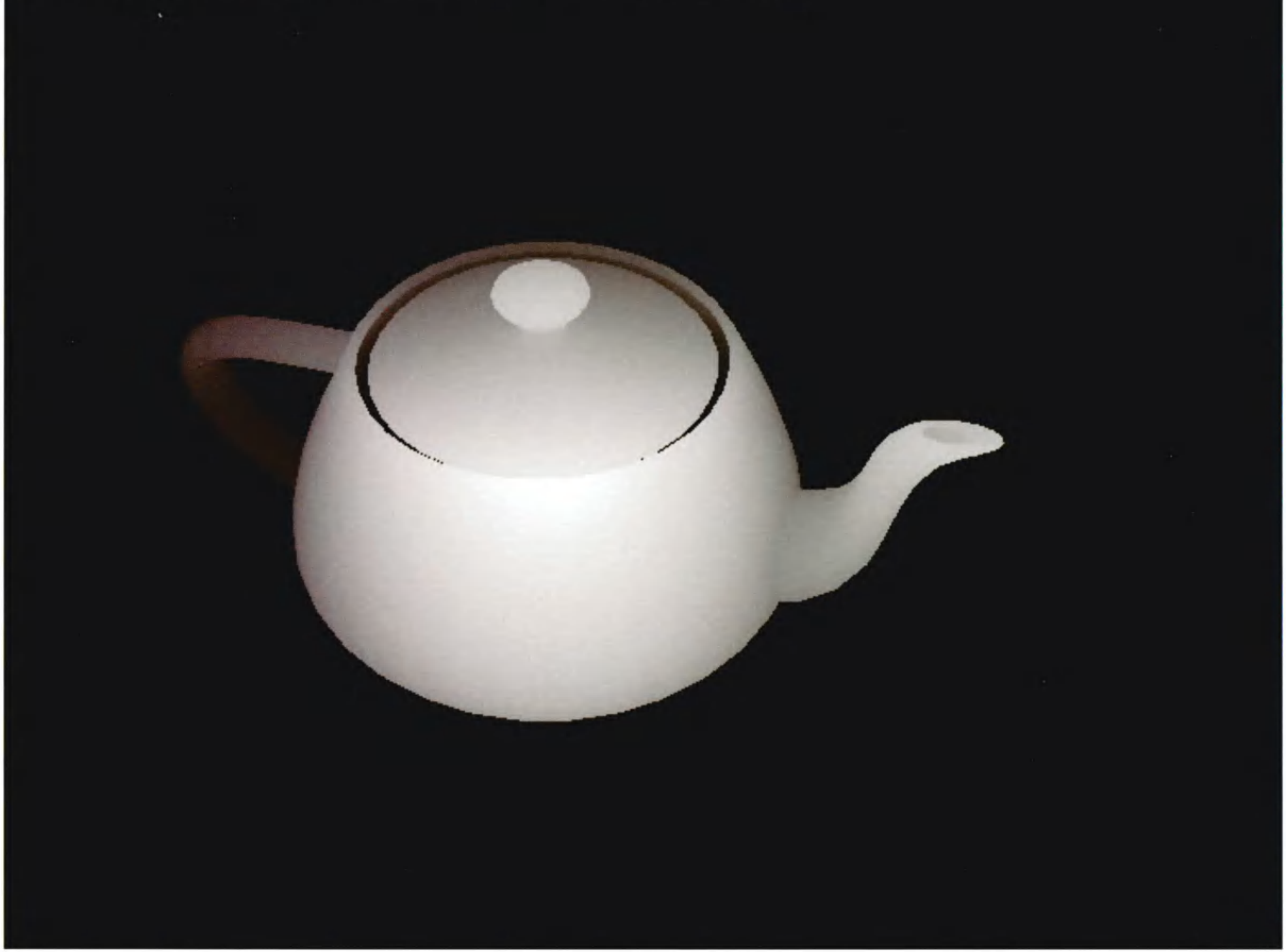
The concept of filtering, or blurring the depth image, is useful in practical stereogram creation. Perhaps the easiest way to understand the purpose of this is to consider a simple example.

Let's assume that you've got a grayscale depth image that is all black, except for a gray square in the middle of the image. For the sake of argument, lets say that the grayscale value of the black pixels is zero, and the grayscale value of the gray pixels is nine.

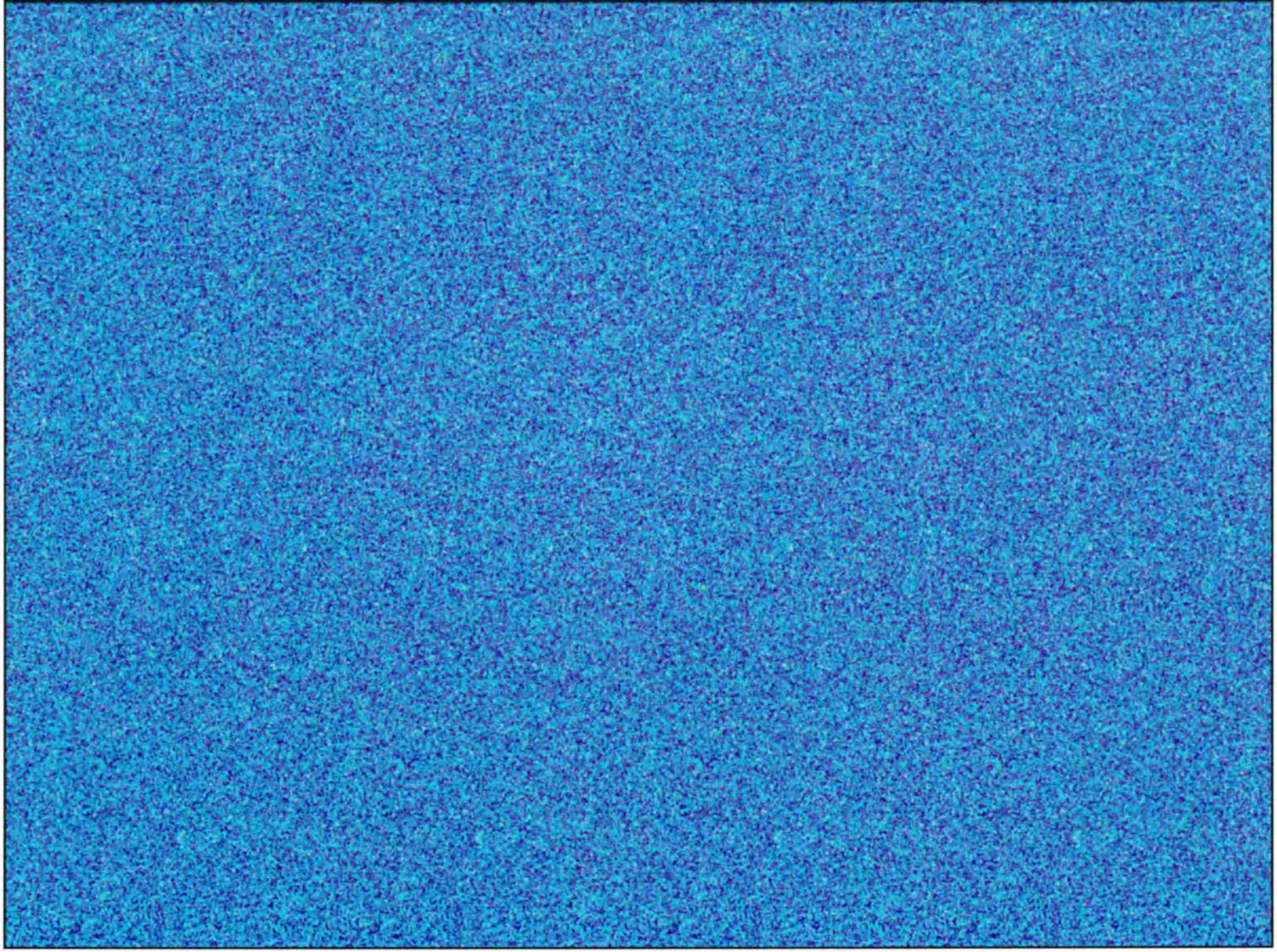
If we create a stereogram with this depth image, the result is a hidden square that is lifted from the wall background. The amount of lift is dependent on a number of things, as I've discussed in earlier chapters. Let's assume that the stereogram is generated using the simplest mechanism, where there is no scaling of the depth image intensity values. If this is the case, the square is created by deleting nine elements from the current pattern to create the left hand edge of the square, and by inserting nine elements into the pattern to create the right hand edge. The resulting stereogram has a total of two levels of apparent depth: the background wall and the square.

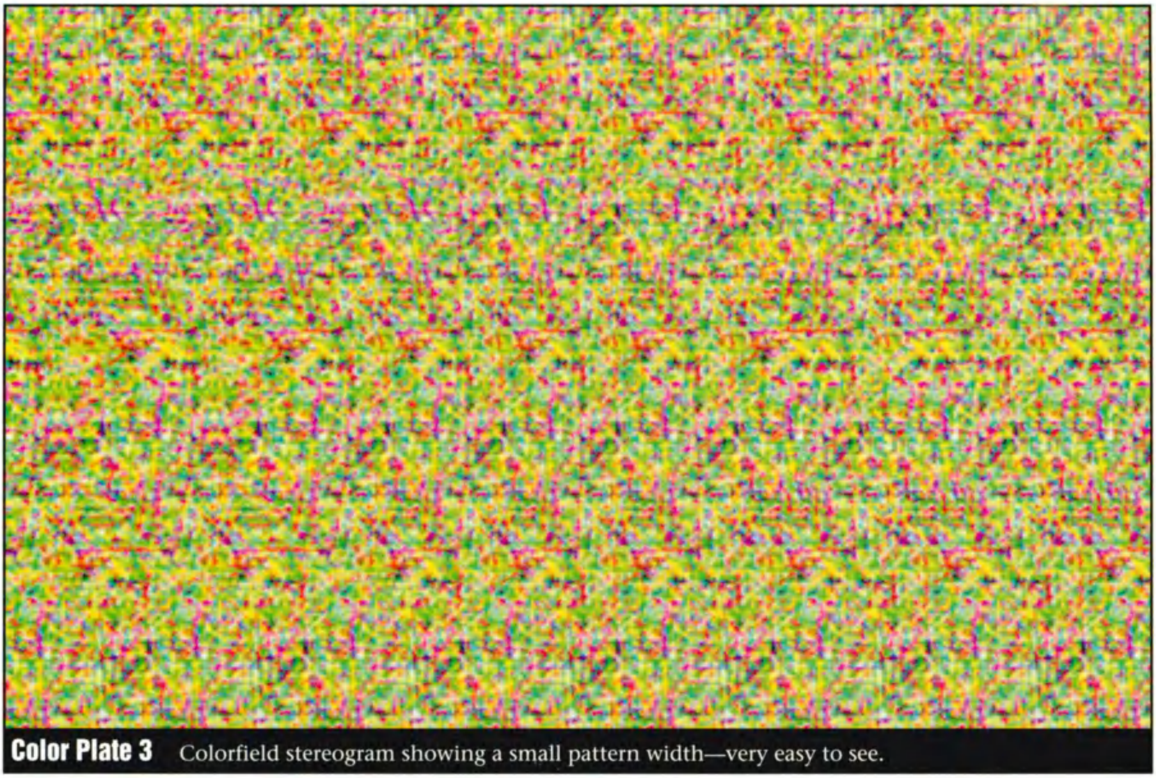
What if, before creating the stereogram, I smoothed this depth image, blurring it so that the edges of the gray square gradually changed to black. In other words, I got rid of any abrupt changes in intensity. Before filtering, one line of the depth file might have grayscale values that look like this:

Color Plate 1 The "Utah Teapot" rendered by the ray tracer POLYRAY, by Alexander Enzmann.

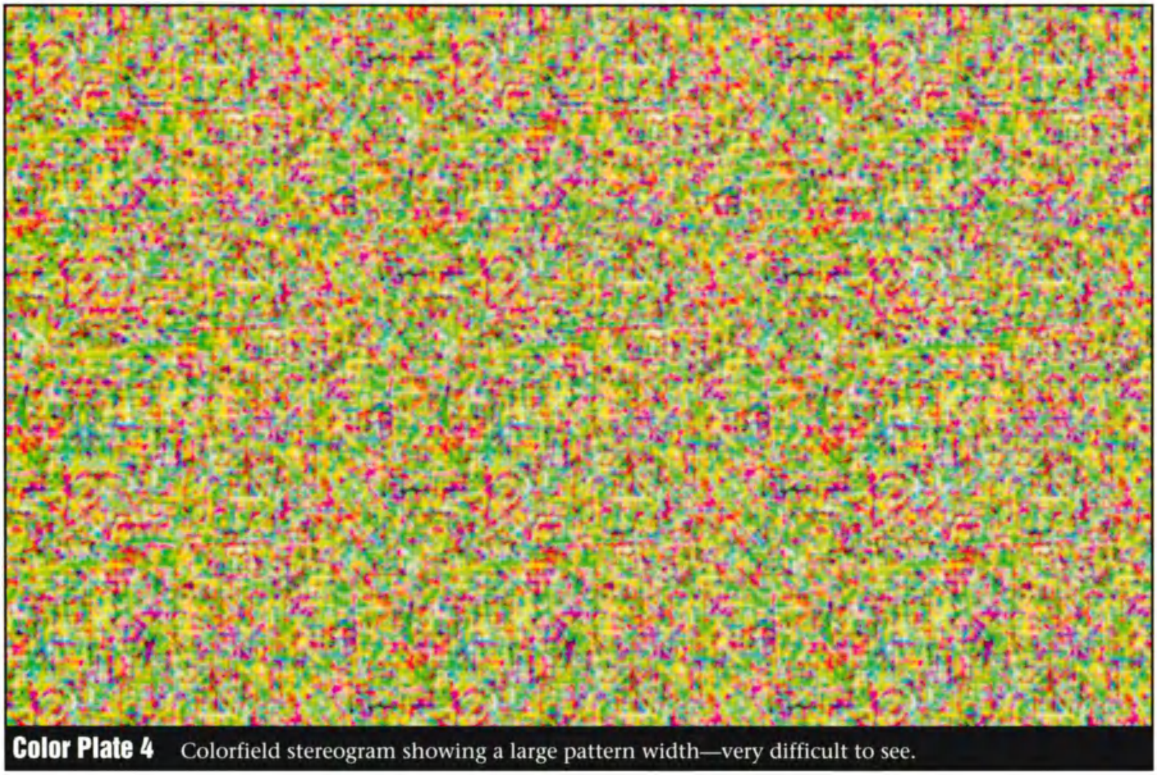


Color Plate 2 A simple, multicolored RDS of the Teapot.

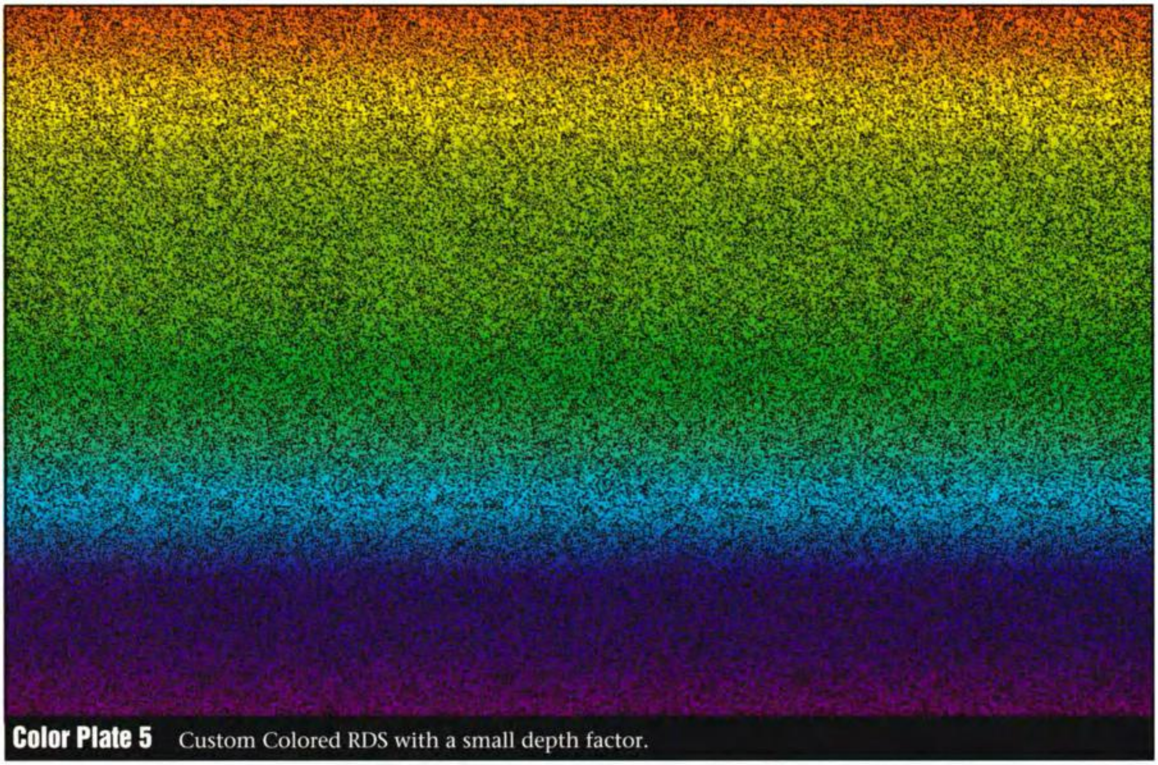




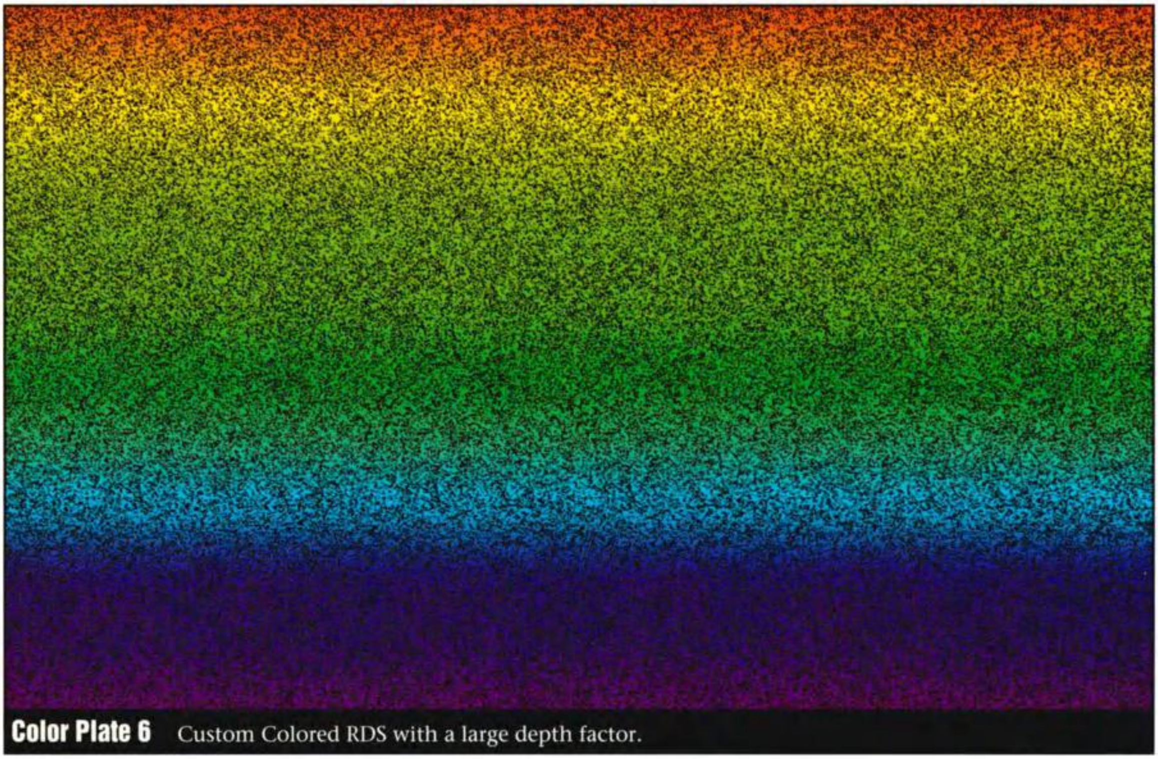
Color Plate 3 Colorfield stereogram showing a small pattern width—very easy to see.



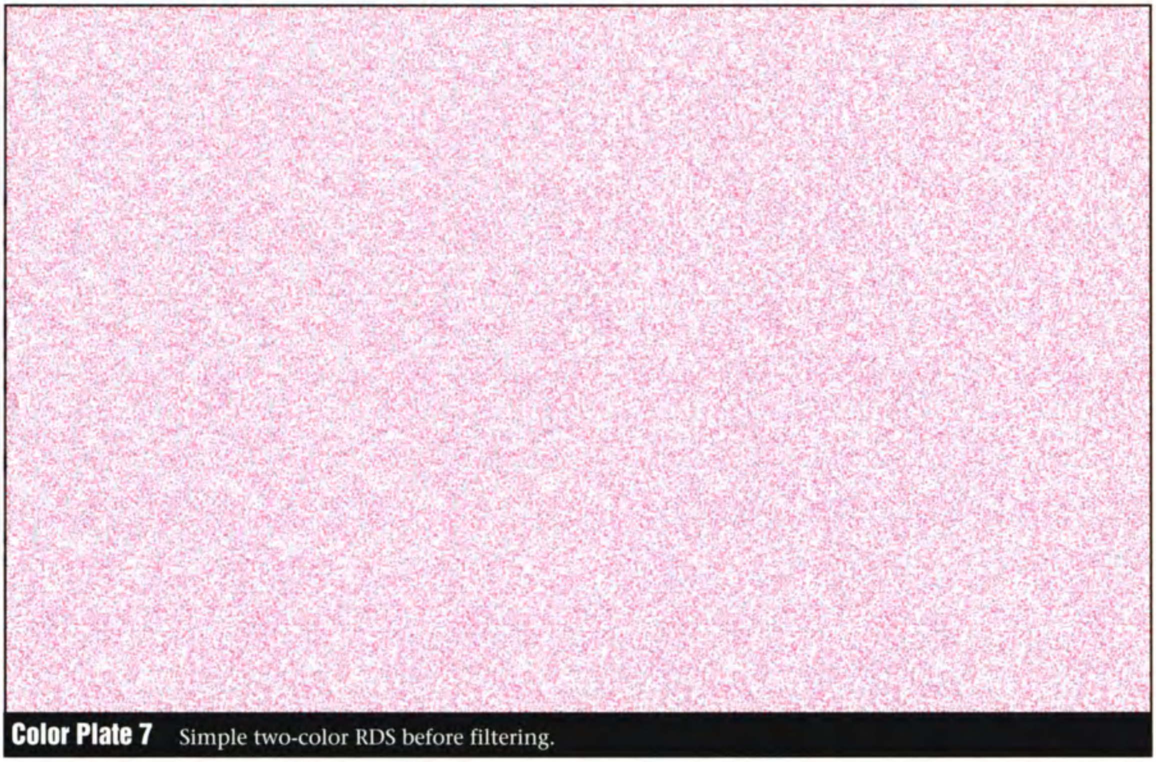
Color Plate 4 Colorfield stereogram showing a large pattern width—very difficult to see.



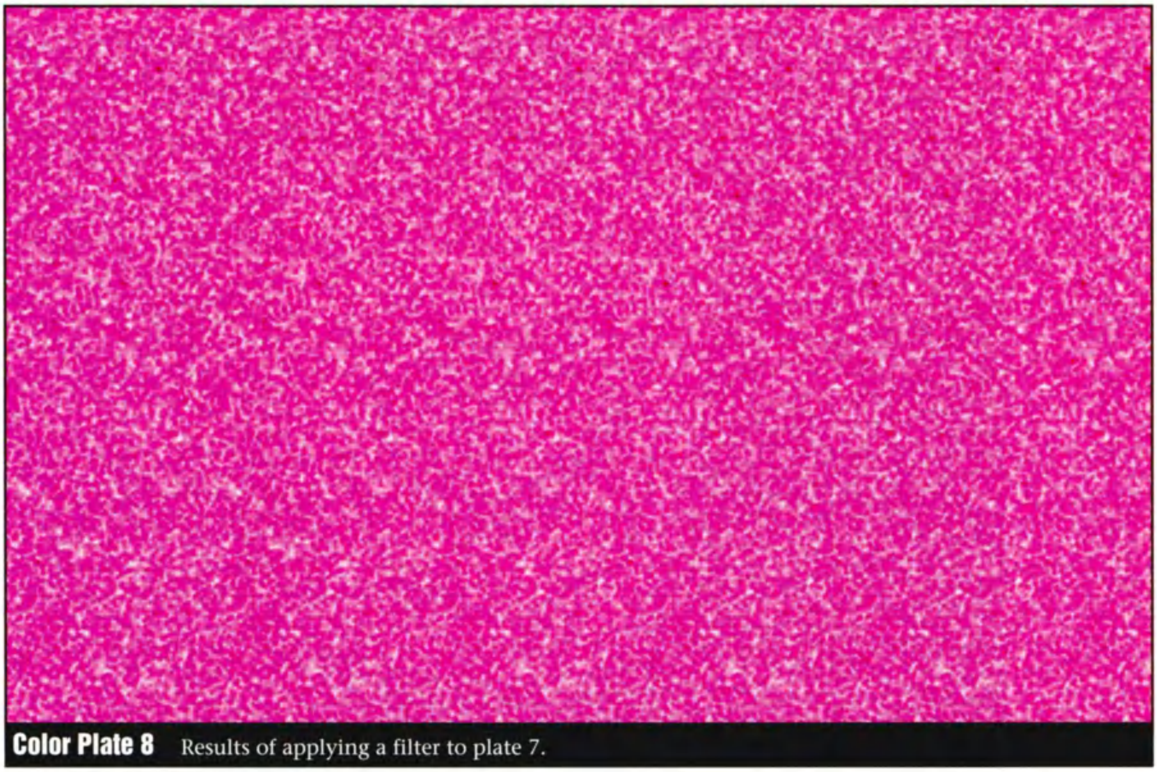
Color Plate 5 Custom Colored RDS with a small depth factor.



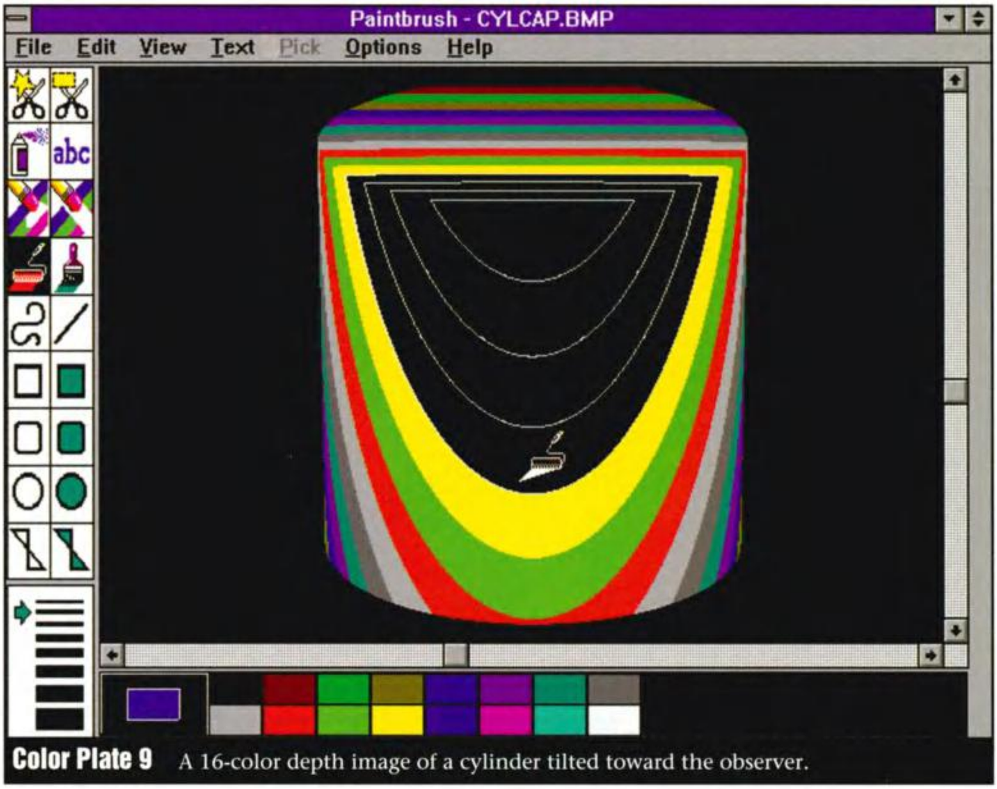
Color Plate 6 Custom Colored RDS with a large depth factor.



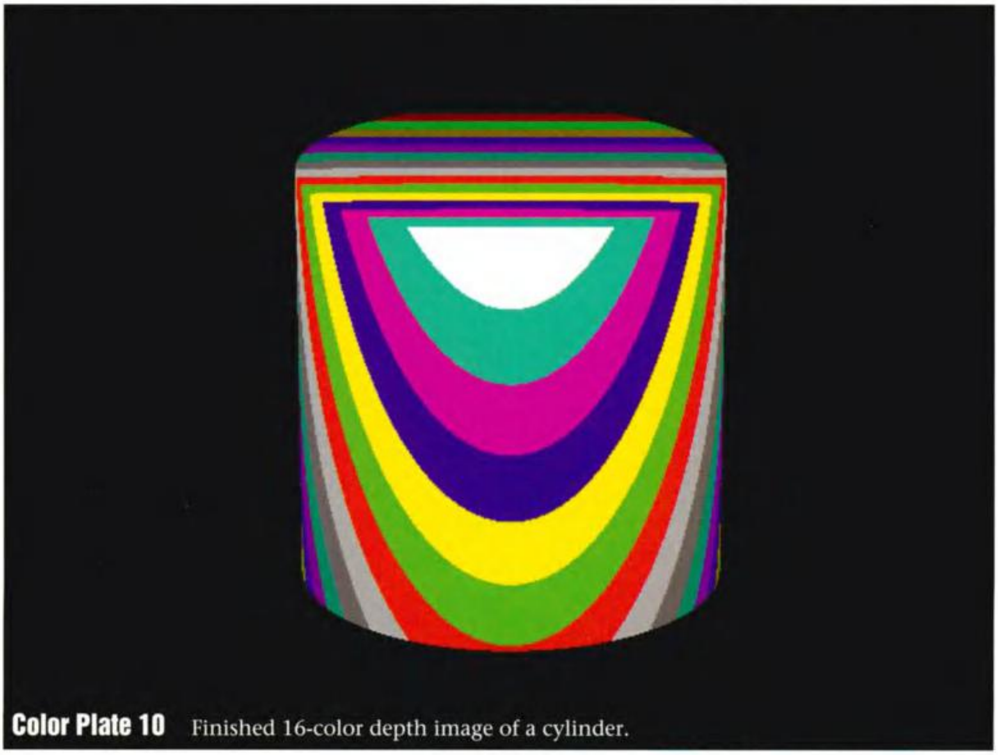
Color Plate 7 Simple two-color RDS before filtering.



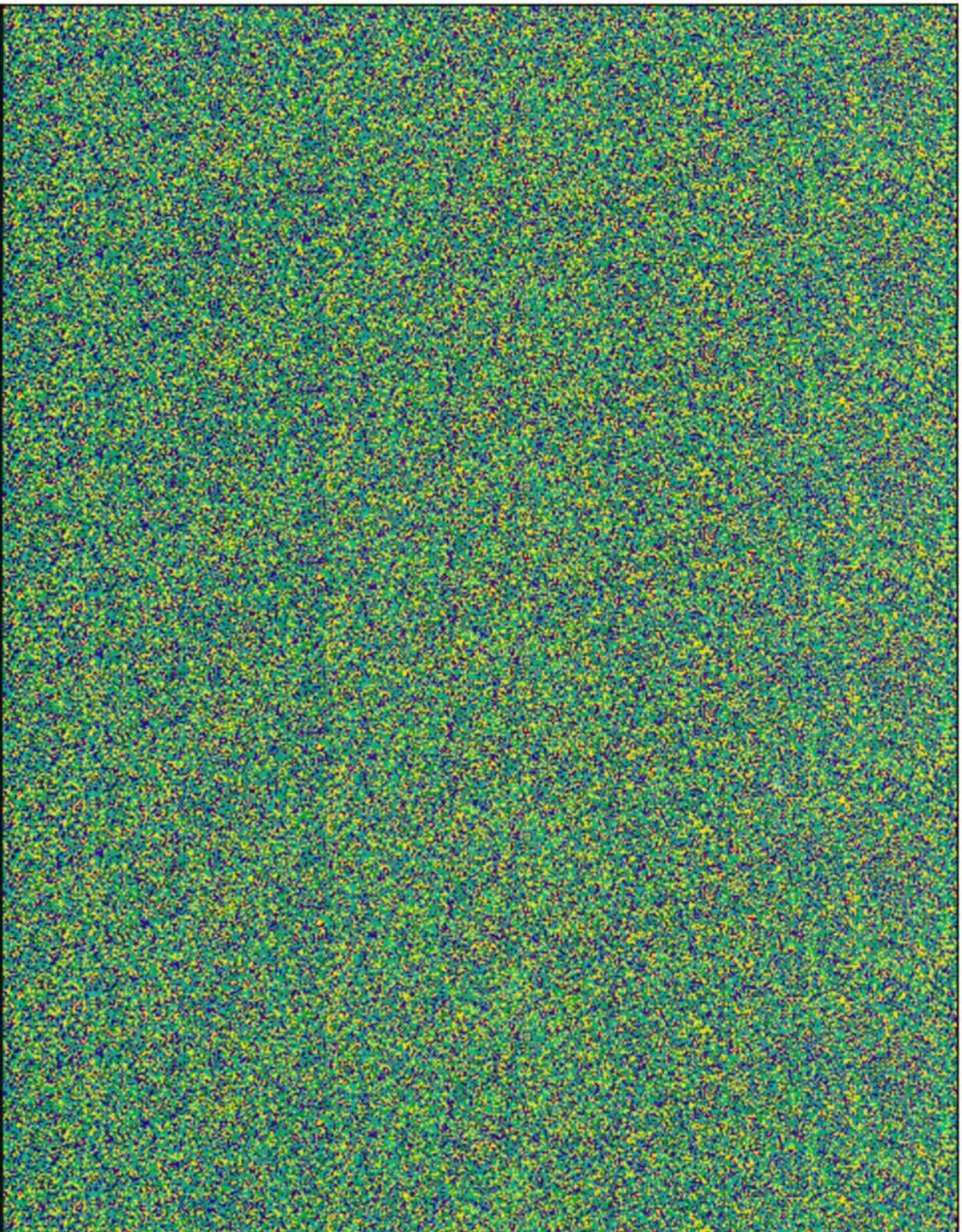
Color Plate 8 Results of applying a filter to plate 7.



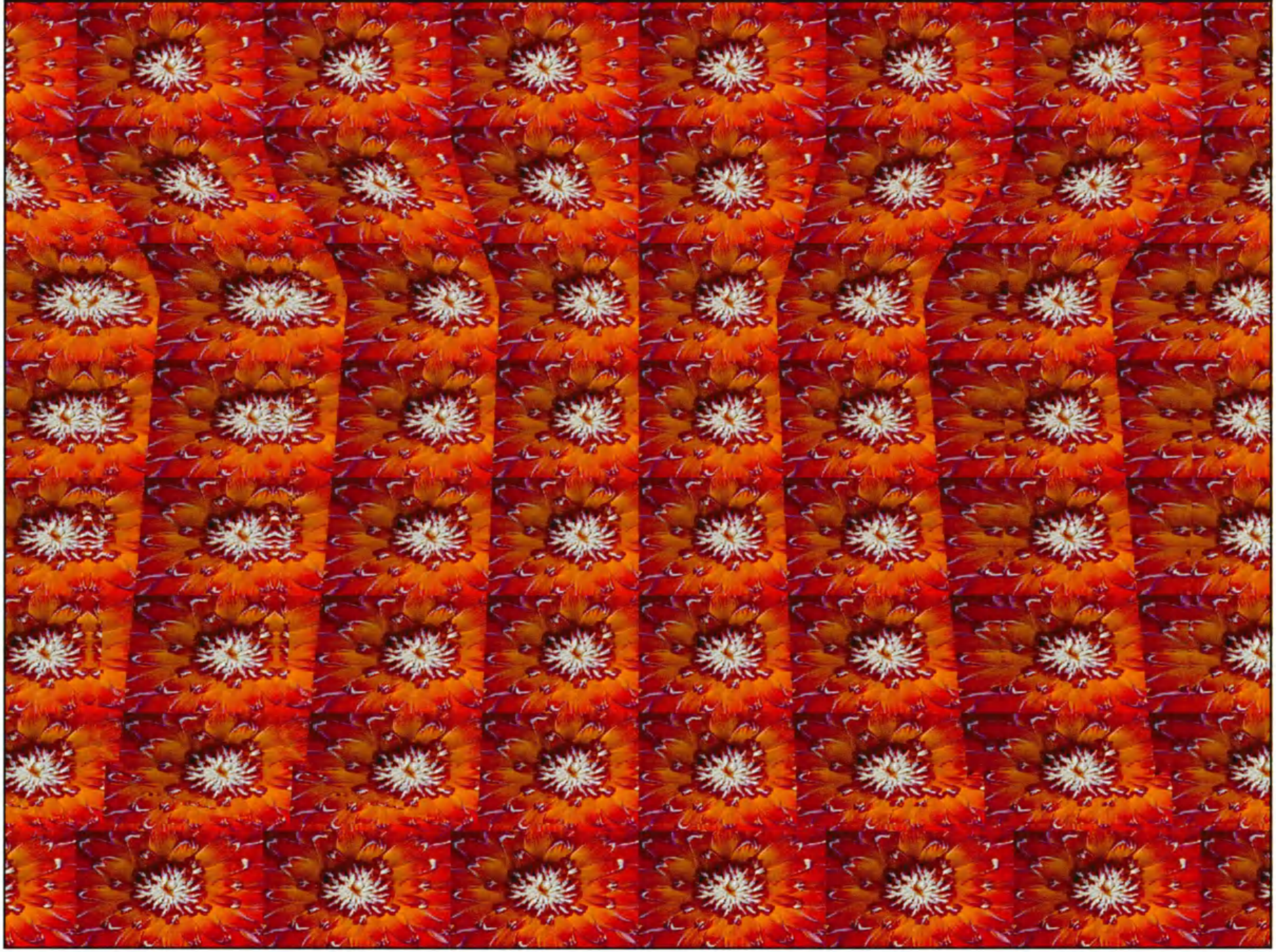
Color Plate 9 A 16-color depth image of a cylinder tilted toward the observer.



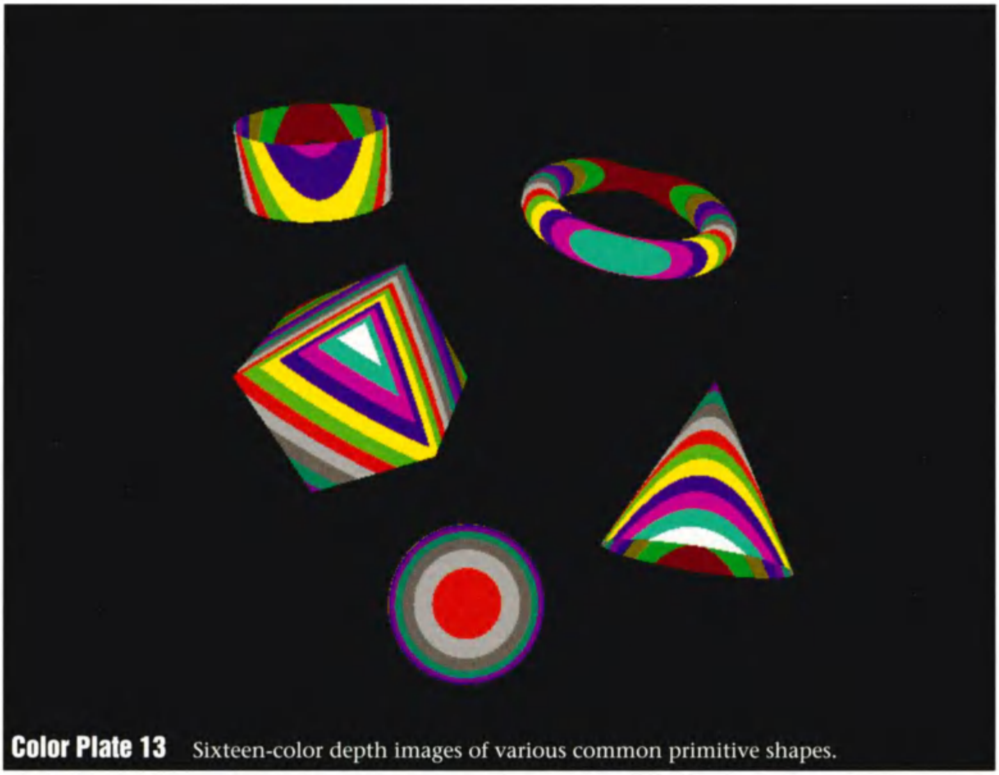
Color Plate 10 Finished 16-color depth image of a cylinder.



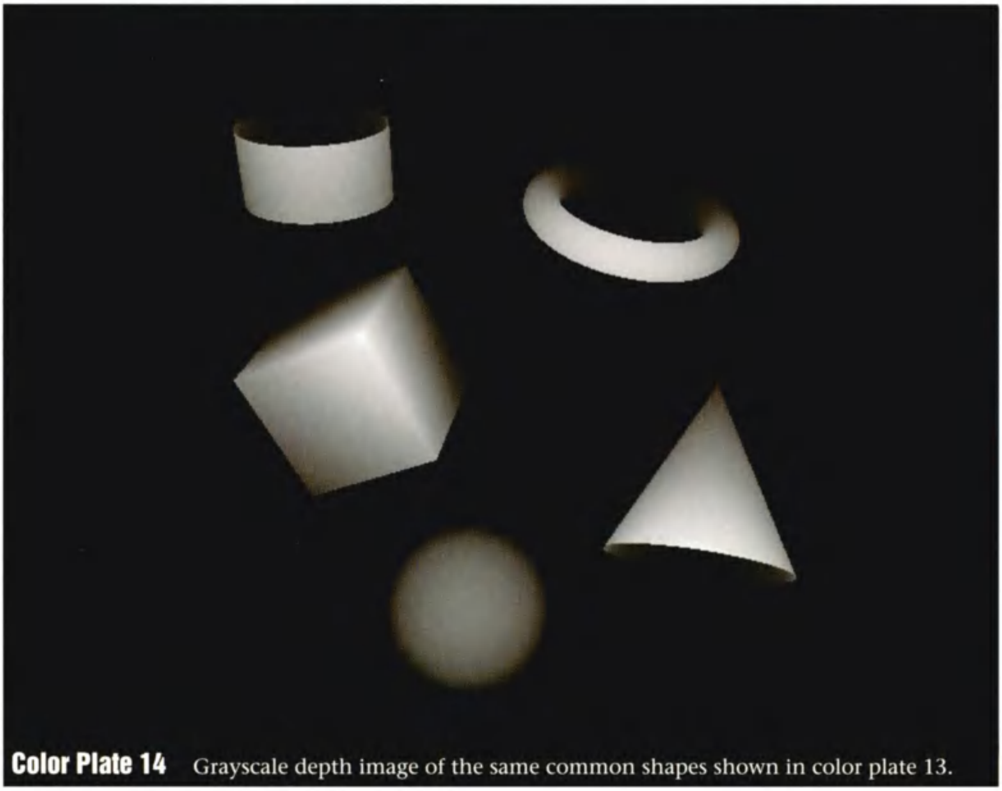
Color Plate 11 Sixteen-level RDS of a cylinder created using POPOUT-LITE.



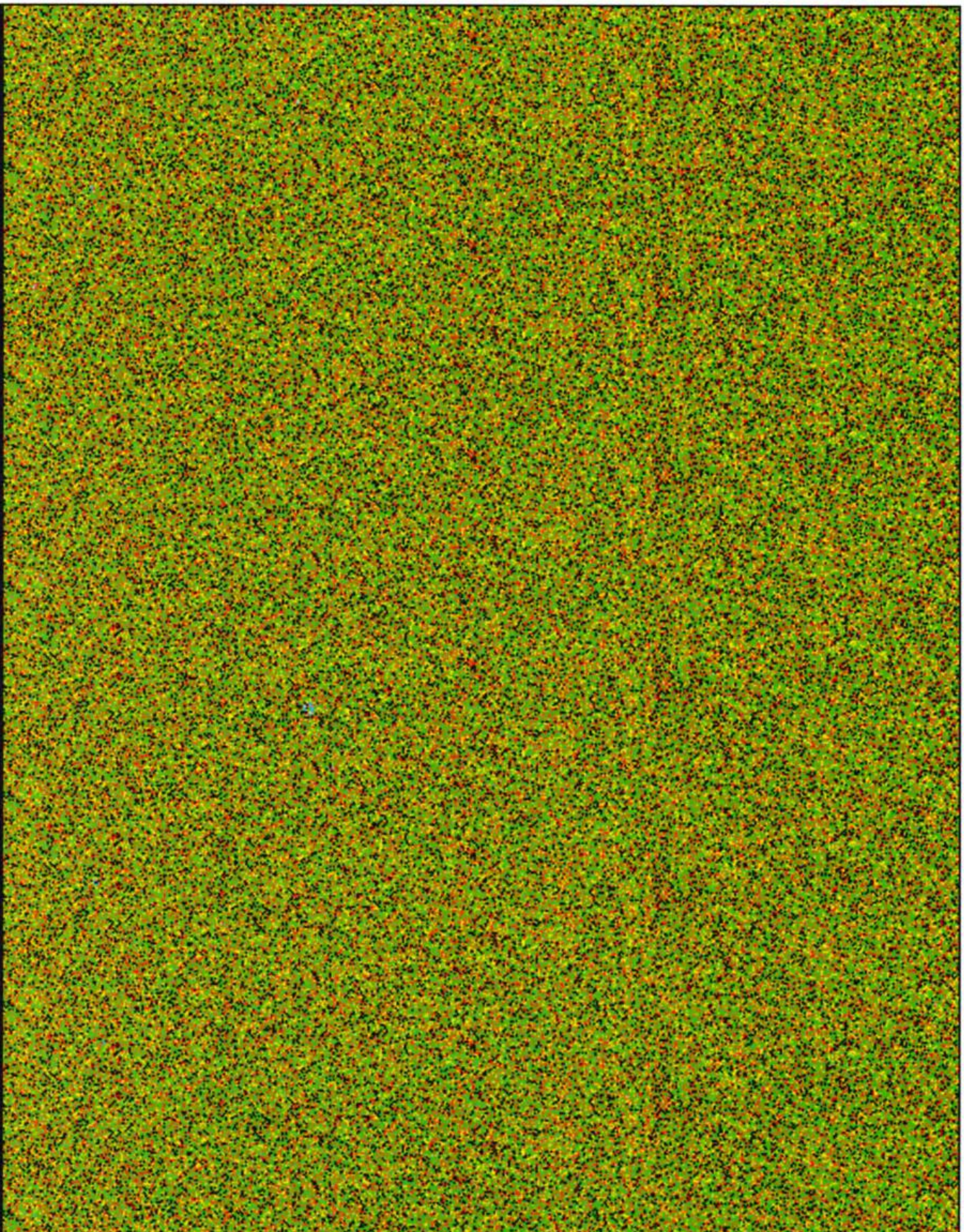
Color Plate 12 Colorfield stereogram of a cylinder containing many levels.



Color Plate 13 Sixteen-color depth images of various common primitive shapes.

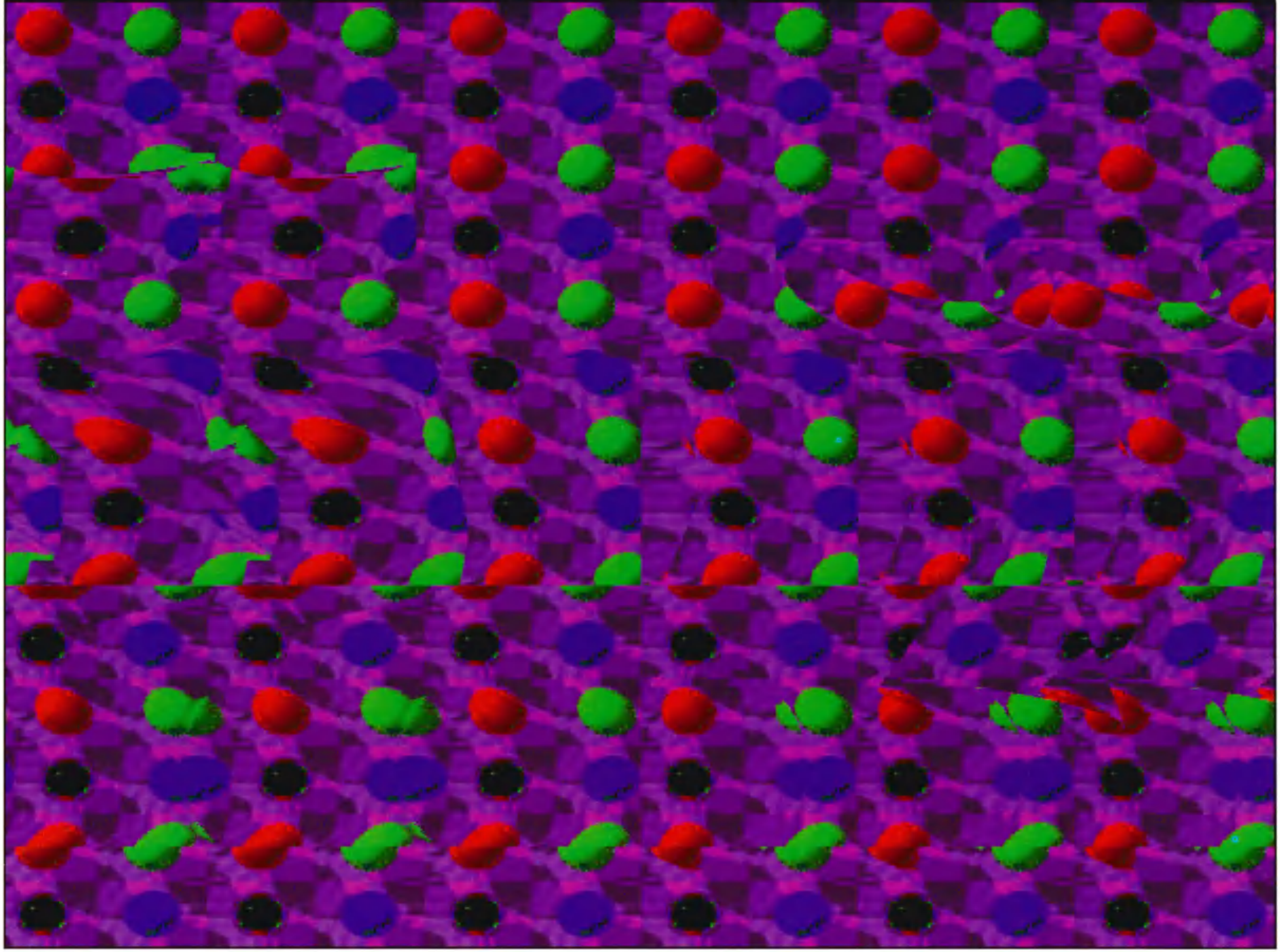


Color Plate 14 Grayscale depth image of the same common shapes shown in color plate 13.

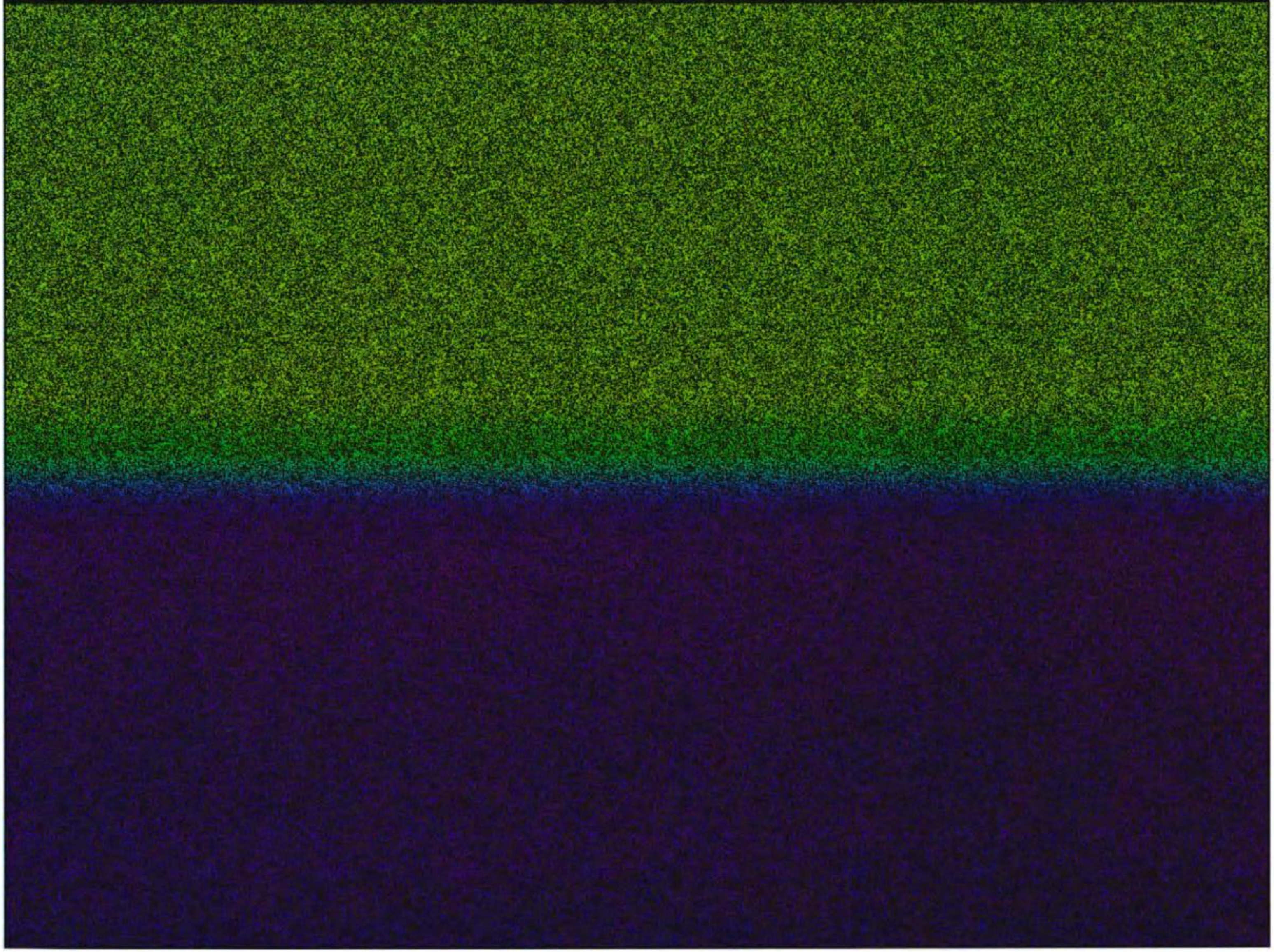


Color Plate 15 Sixteen-level RDS containing various primitive shapes created with POPOUT-LITE.

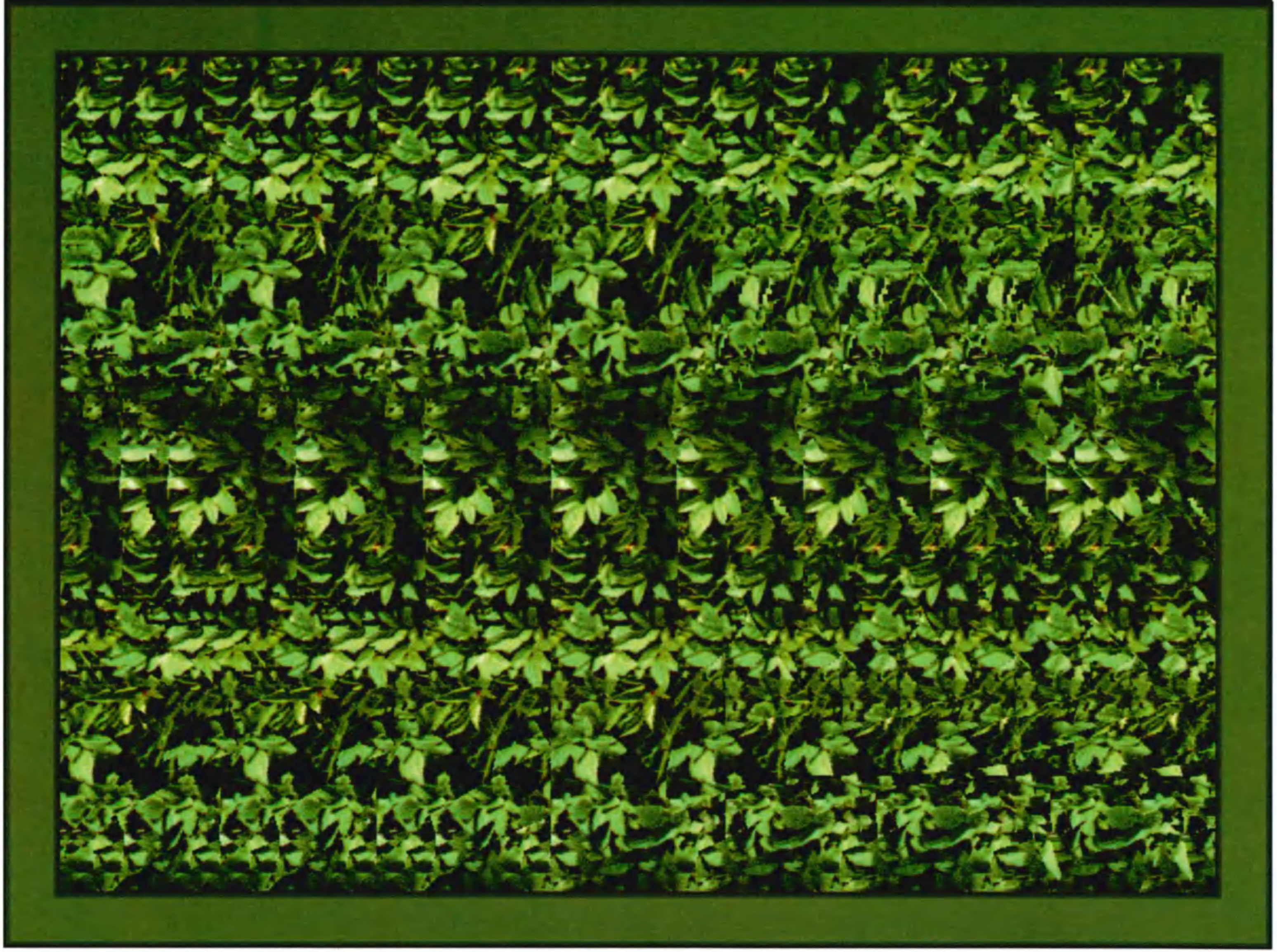
Color Plate 16 Colorfield stereogram containing the same shapes as in color plate 15, with many more levels.



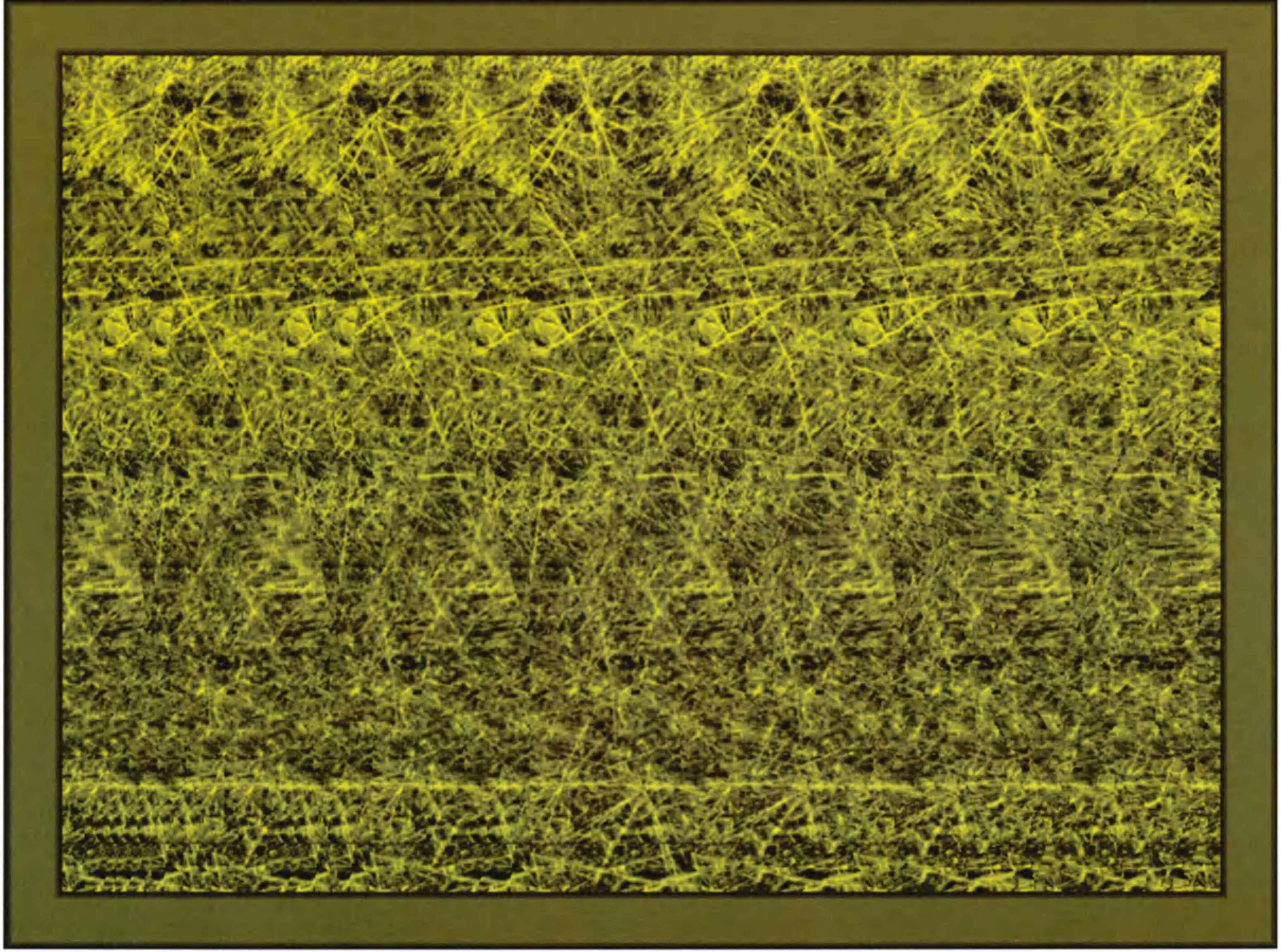
Color Plate 17 Many-level RDS containing five men walking in a circle. These people figures were designed by Will Wagner.



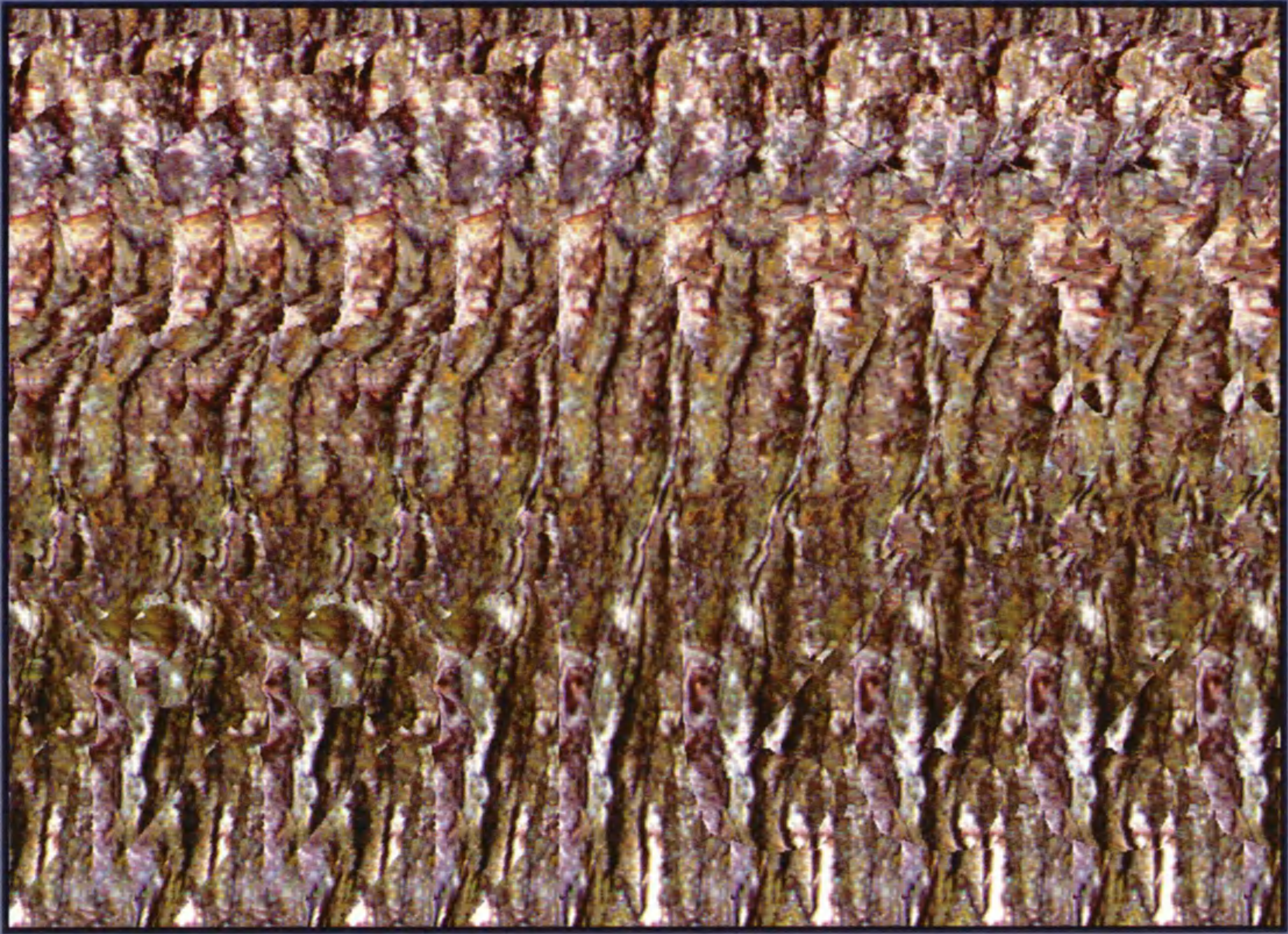
Color Plate 18 “Butterfly” Copyright © 1994 Chaos FX Inc., Milton, Ontario



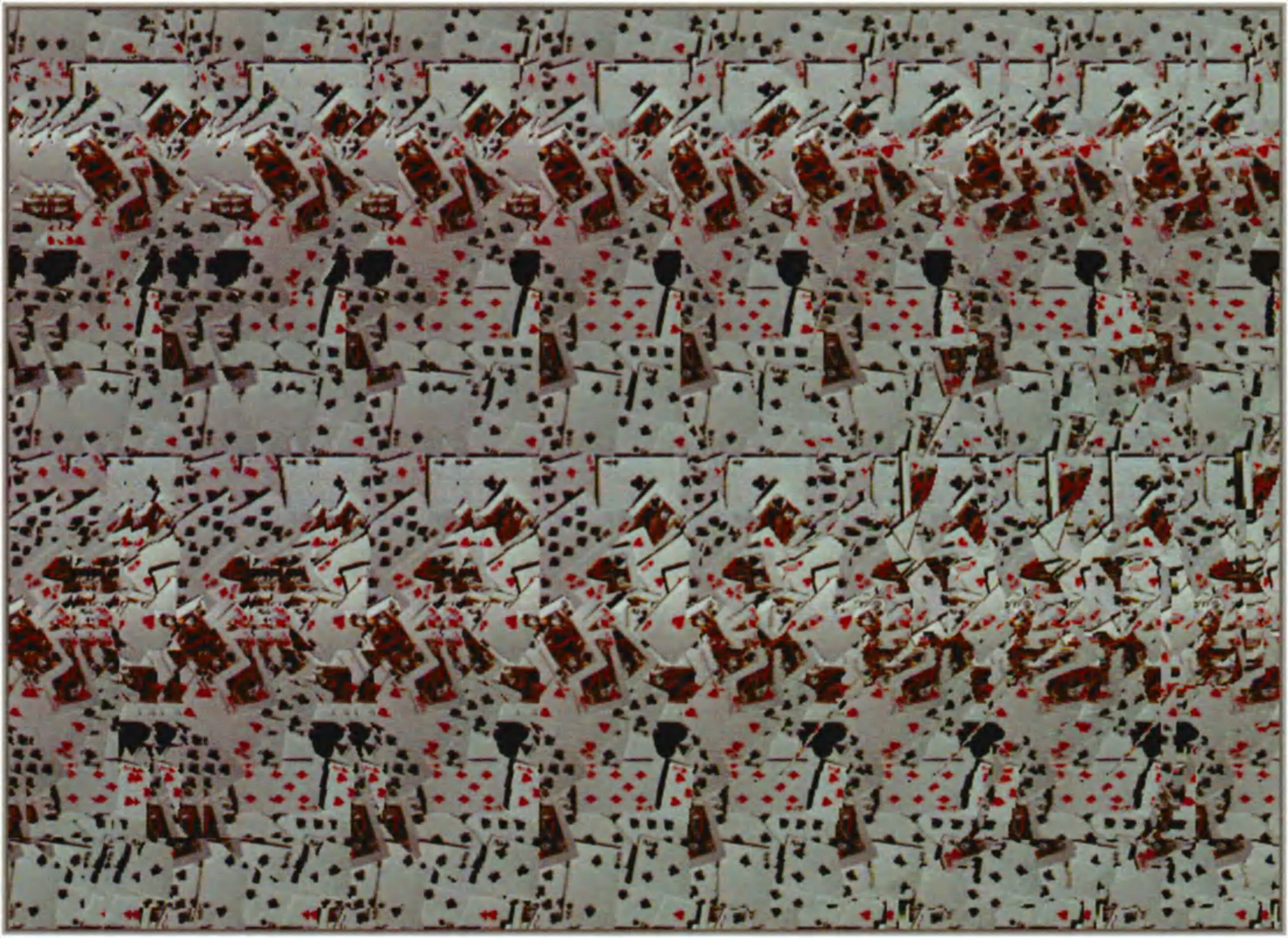
Color Plate 19 “On the Prowl” Copyright © 1994 Chaos FX Inc., Milton, Ontario



Color Plate 20 “Out on a Limb” Copyright © 1994 Chaos FX Inc., Milton Ontario


]

Color Plate 21 "Joker's Wild" Copyright © 1994 Chaos FX Inc., Milton, Ontario



```
0000000000000000000009999999999990000000000000000000000000
```

As you can see, the above row of numbers represents the grayscale values of a line in the depth image that runs through the square.

After filtering the square, this same line might look something like this:

```
0000000000000000000135799999999753100000000000000000000000
```

Do you see how the pixels in this line would start at black, and then gradually grow to having a value of 9? The left-hand side of the square would be blurry. The same thing goes for the right-hand side, as well as the top and bottom.

So what impact does this have on stereogram generation? The first thing you may notice is that the stereogram that results from this example will have six levels of depth, instead of just two. Instead of looking like a square sitting in front of a wall, the image will look more like a rubber wall, where a square is being pushed up against the wall from behind.

You may have noticed in all the examples that I've used with numbers, I've chosen a starting pattern width that contains ten elements. There is no special reason for this except that in this particular example, a ten-element starting pattern really isn't long enough; deleting nine elements from the pattern leaves a pattern width of one element. A good rule of thumb is to start with an initial pattern from which you'll never delete more than half of the elements. With this example, I'd start with a pattern width of about 20 elements.

This kind of filtering can be done when the depth image is created, and doesn't necessarily have anything to do with the stereogram software. For some images, this elimination of abrupt changes in depth can be desirable and add some interesting effects.

TIP: This filtering technique works especially well when creating colorfield stereograms. Abrupt changes in depth file intensity usually results in an unpleasant image that is difficult to view (see fig. 6.4).

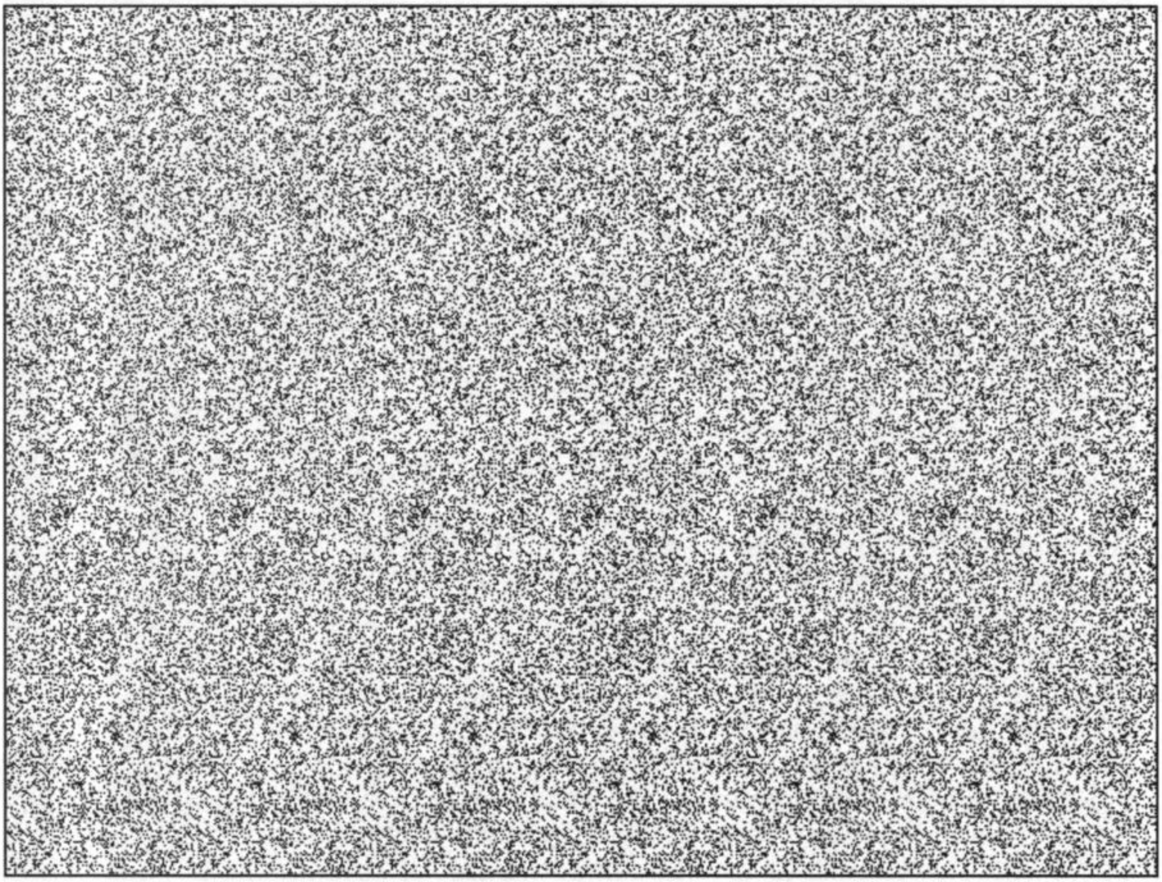


Figure 6.4

No filtering of the depth image.

There is a second instance where depth file filtering can improve the quality of the image, but this is dependent on the stereogram software. When creating large images, I showed you earlier that it's fairly essential that the stereogram software has the ability to magnify the depth image as the stereogram is processed. Otherwise, you're always limited to creating stereograms that are the same dimensions as your depth image.

If the stereogram software filters the depth image as it enlarges it, the result is a higher quality stereogram. This is because the edges of the individual magnified levels tend to merge together more than if the depth image is enlarged simply by replicating the pixels.

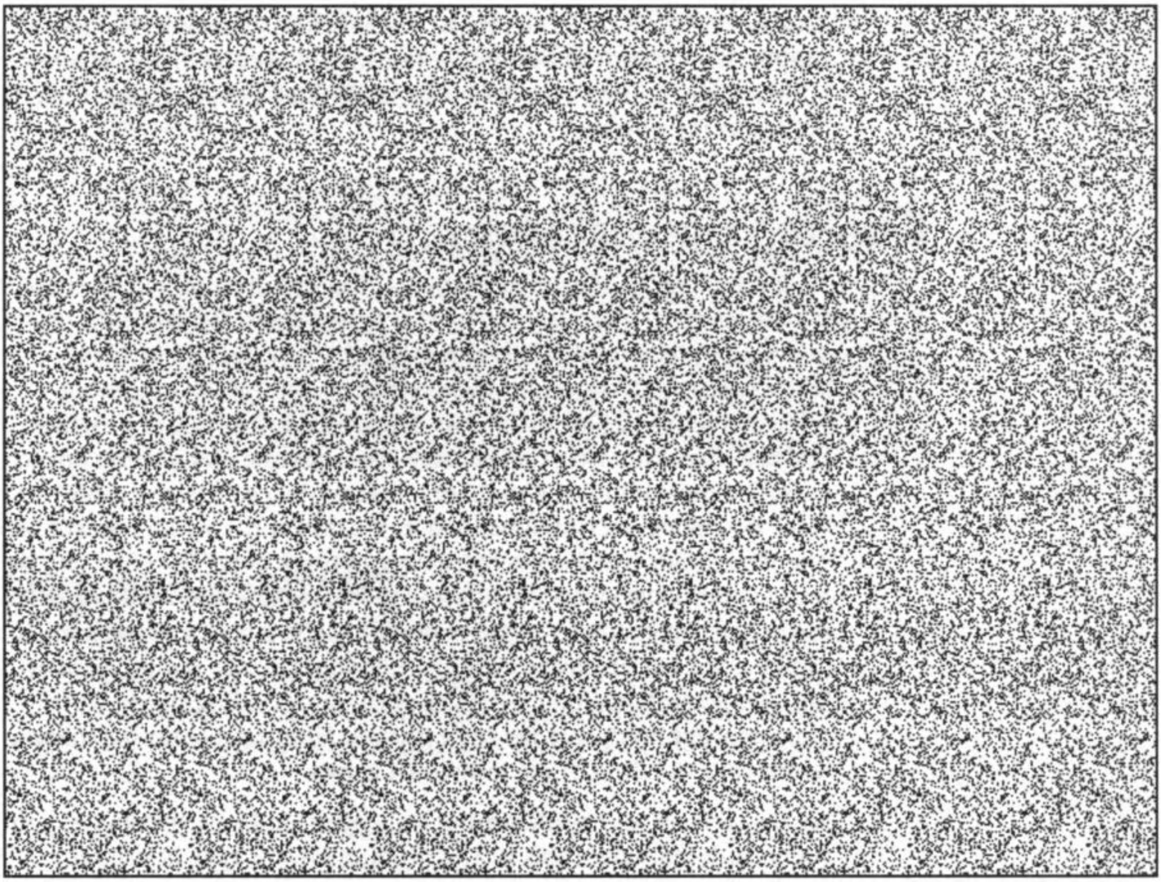


Figure 6.5

Depth image was filtered first.

Figure 6.4 is an example of a stereogram in which the depth image has some very abrupt changes in intensity. In figure 6.5, I start with the same depth image, but filter it before creating the stereogram. Notice the difference?

A number of specific filters can be used to achieve these affects, all of which are beyond the scope of this book. What I really want you to understand is that if you've got a depth image that doesn't have a lot of depth, but does have abrupt changes between the different grayscales, filtering can help to create some interesting images. This filtering can be done with a number of available programs and typically doesn't necessarily have to be done by the stereogram software.

Because POPOUT-LITE is a simple, 16-level RDS generator, it doesn't support the concept of filtering or blurring the depth image. Figure 6.5 was created using POPOUT-PRO, which has a depth image filter built into the program.

Filtering the Stereogram

After a stereogram is created normally, you can create some interesting effects by blurring the stereogram image itself. For example, let's say you've just created a stereogram consisting only of black-and-white pixels. This stereogram can be passed through a smoothing filter that tends to blur the image. The white dots won't be as white, and the black dots won't be as black. The amount that a white dot is darkened depends on the number of nearby black dots; the more black dots there are near a white dot, the darker the white dot will be. The same holds true for the lightening of black dots. The effect is really appreciable when the filtering mechanism is also used to resize the image. For example, if you're doubling the size of a stereogram, it can be done not just by replicating individual pixels, but by calculating new pixels based on the values of the surrounding pixels. This same concept holds true for color images. Filtering a color stereogram results in a blending of the individual colors.

Because a black-and-white printer only prints in black and white, you really can't appreciate this sort of filtering on this type of printer. Some color printers, such as those with a dye sublimation capability, can print the results of this kind of filtering. Also, your PC monitor can certainly show the results of stereogram filtering. You can see the effect of filtering a random dot stereogram in color plate 8, in the center of this book. This image was created by first generating a simple, two color stereogram. Color plate 7 is this simple, two color image. I created color plate 8 by using an image processing program, unrelated to the stereogram software, to soften the image of plate 7. Plates 7 and 8 are really a before-and-after example of stereogram filtering.

Filtering a stereogram can be done using any number of graphics programs available, and is therefore probably not a function of your stereogram program.

This is true for both POPOUT-LITE and POPOUT-PRO. Color plate 7 was generated using POPOUT-PRO, but then it was filtered to create plate 8 by using a filtering program unrelated to stereogram generation.

TIP: If you've got access to such a filtering program, you can achieve some interesting effects by selecting different types and widths of filters to use on your stereogram. Some interesting experiments involve determining how much the image can be blurred, yet still remain a legitimate stereogram. You can reach a point where the image is blurred so badly that the hidden image is no longer viewable!

Depth Linearity

I'd like to spend some time discussing and explaining an annoying imperfection that appears in many stereograms. You may have seen this defect in your own stereograms or in others. Many so-called professionally created images fall short of perfection because of this defect.

I'm referring to something called *depth linearity,* or more accurately, *depth nonlinearity*. Depth linearity simply refers to the accuracy in which a stereogram represents equal changes in apparent depth as specified by the depth image.

This implies that the grayscale values of a depth image are an indication of the desired apparent depth of the image hidden within the stereogram. For example, suppose that the following line is a line from a depth file, and the numbers represent the grayscale intensity of the pixels in the line:

```
000000000000000111111112222222333333332222222111111110000000
```

This depth file creates a stereogram with a total of four levels of depth. Where the zeroes change to ones, an element is deleted from the current pattern, creating a raised region. At the point where the ones change to twos, another element is deleted, causing a raised region to be created on top of the first. This happens again at the point where the twos change to threes. Elements are inserted to terminate each of the three raised regions where the threes change to twos, where the twos change to ones, and again where the ones change to zeroes.

Let's call these four levels, level 0, 1, 2, and 3, which correspond to the depth image grayscale values that were used to create them. When viewing the resultant stereogram, each of these four levels appears to be a certain distance away from the observer. Level 0 is the farthest, and level 3 is the closest.

Just how far away do these four levels appear to be? Furthermore, what is the apparent distance between level 0 and level 1? How about the distance between level 1 and level 2, or the distance between level 2 and level 3?

Most artists, when creating a stereogram depth image, assume that for each unit of change in the depth file pixel intensity value, there is a fixed unit of change in the apparent depth. In other words, the apparent distances between each of the four levels above is the same.

Well, that's not the way it works! The best way to explain this might be with a sketch. Figure 6.6 is a sketch of an observer staring divergently at a stereogram. I've left out some things to avoid a lot of clutter. For example, instead of showing the actual pattern, I've just indicated a distance called *P* to represent the current pattern width.

I've also marked some other distances in figure 6.6. *E* is the distance between the observer's two eyes, and *D* is the distance between the observer and the image. *A* is the apparent depth of the current pixel that the observer is staring at. The observer sees the current pixel at the point where the two imaginary lines of sight cross.

Using simple geometry, I can calculate the apparent depth of the pixel by using the rule of similar triangles (because the triangles are proportional). This means that the triangle which has A as the height and P as the base is just a smaller version of the triangle that has (A+D) as the height and E as the base. Equation 6.1 shows this relationship.

Equation 6.1 Stereogram Distance Relationships

$$A / P = (A + D) / E$$

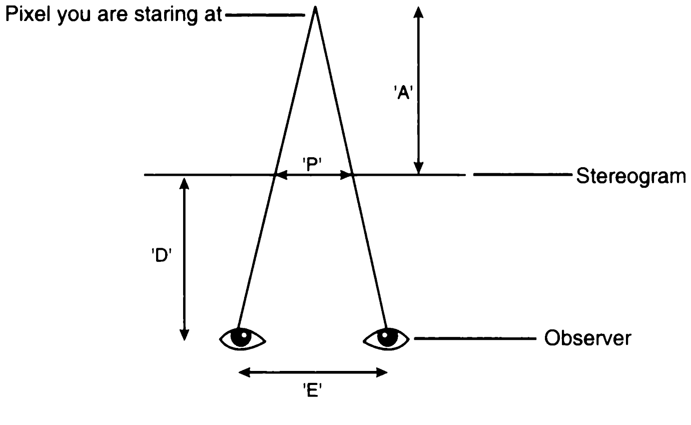


Figure 6.6

The general case of divergent viewing.

If I multiply both sides of the equation by P*E, the result is:

A * E = (A + D) * P

Then, if I multiply out the right-hand side, I get:

A * E = (A * P) + (P * D)

If I then subtract (A * P) from both sides, the result is:

(A * E) – (A * P) = (P * D)

Now, I can factor the A out of the left-hand side for:

A * (E – P) = (P * D)

Finally, if I divide both sides by (E – P), I've solved the equation for A. I've shown this in equation 6.2. You may notice that if E = P, this division is not valid, because I'd be dividing by zero. What does it mean when E = P? It means that the pattern width is the same distance as the distance between the observer's eyes. This would place the point of focus infinitely far away from the observer, and the result would be a nonviewable stereogram. This is the reason that the maximum pattern width must be somewhat smaller than the distance between the observer's eyes.

Equation 6.2 Equation to Calculate Apparent Distance

A = (P * D)/(E – P)

Equation 6.2 allows me to calculate the actual apparent distance of the pixel; that is, how far the pixel appears to be behind the plane of the image. If I know the distance between my two eyes, the distance between me and the stereogram, and the current pattern width, I can calculate the apparent distance of any pixel that I'm staring at.

In an earlier chapter, I mentioned that the apparent depth in an image is in part determined by the current pattern width. As you can see from the above equation, it is also a function of the distances D and E.

I know that the distance between the observer's two eyes certainly will never change. In other words, E is a constant value for any one person. Also, I'm going to assume that the observer isn't moving his or her head back and forth while trying to view the image, which is pretty tough to do anyway. This means that the distance between the observer and the image, distance D, is also a constant. The only remaining variable in the equation is P, the current pattern width.

In short, if both E and D are constants, the apparent depth of the pixel reduces to being a function only of the pattern width. You can see from equation 6.2 that as P increases, P*D increases, while E–P decreases, so the apparent depth A will increase.

Look at figure 6.6 and imagine P increasing in width. With D and E fixed, the lines of sight will spread apart as P increases, so that the point of focus will be farther behind the image. In other words, as the pattern width increased, so did the apparent depth of the pixel.

"So what's the point of all of this?," you ask. After all of that, I'll tell you that I'm really not interested in calculating the actual apparent depth of a pixel. What I am interested in is how this apparent depth changes as a function of the pattern width.

To stay within the scope of this book, I'm going to simplify things and include a graph that shows how equation 6.2 can be plotted. I'm going to do it in such a way as to ignore the D and E constants, and just compare how apparent depth changes with pattern width.

Figure 6.7 is such a graph. The curved line represents the actual relationship between apparent depth and pattern width. The straight line represents how I wish it really was! For every unit of depth image intensity change, I would like to have a corresponding unit of change in apparent depth, as this makes life a lot easier when creating depth images.

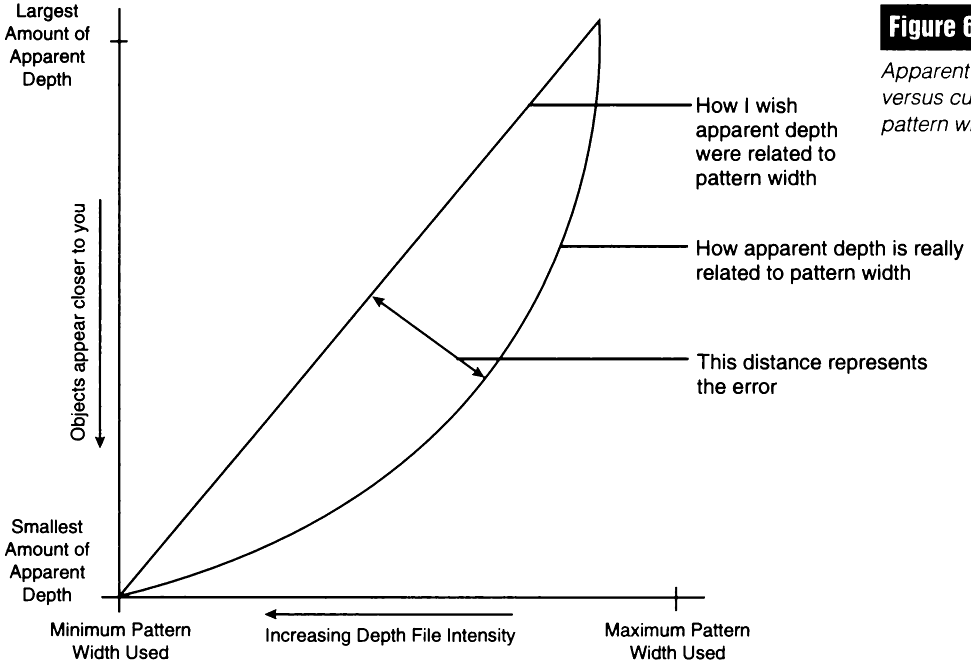


Figure 6.7

Apparent depth versus current pattern width.

The horizontal line at the bottom of the graph represents the different current pattern widths that are used in a stereogram. The left-most point of this line represents the minimum width of the pattern, while the right-most point represents the maximum width of the pattern. The vertical line measures apparent depth. The bottom of the vertical line represents the minimum apparent depth in the image, and the top represents the maximum depth.

As you can see from the figure, the apparent depth increases as the current pattern width increases. The problem is that this increase isn't linear. If the current pattern width is doubled, the apparent depth doesn't double as it would if it followed the straight line.

Simply put, the problem is that because the apparent depth is not a linear function of the current pattern width, it's not a linear function of the intensity of the depth image either. The result is distortion in the linearity in the apparent depth of the image.

You may have noticed that I didn't put any numbers on the graph in figure 6.7. The reason is that to put actual numbers on it, I no longer can ignore the constants D and E, but must assign some real numbers to them. The purpose of figure 6.7 is simply to show you that the apparent depth to pattern width relationship deviates from an ideal straight line. The amount of that deviation depends on the difference between the maximum and minimum pattern widths, as well as the values of E and D.

If E and D are fixed, the amount of deviation from the ideal line is a function of the difference between the maximum and minimum pattern widths in the image. As this difference increases, so does the deviation. As this difference decreases, the curved line in the graph moves closer and closer to the ideal line, and the distortion is less noticeable.

What this means in practice is that the effect is really only noticeable when you've got an image with a lot of depth in it, meaning that you're using a large depth factor, and a significant amount of the original pattern width is sacrificed for depth. Figure 6.8 has a lot of depth. It was generated with a 60 percent depth factor.

Figure 6.8 is really two stereograms that I cut and pasted together. The top half of the image was generated by the normal method, where the number of element deletions and insertions was a simple function of the depth image pixel intensity. The bottom half was generated by taking into account this nonlinear relationship between apparent depth and pattern width, and by telling the stereogram software to correct for it.

Unfortunately, most stereogram software does not correct for nonlinearities in apparent depth. The next time you're browsing through some stereogram posters, see if you can pick out instances of this nonlinearity. Notice especially if the image has a floor or side wall. If the floor tends to bend down, or if the side wall tends to bend outwards, this is an indication of depth nonlinearity.

With stereograms that contain only a few levels, the inclusion of depth linearity correction would be an extreme overkill. For this reason, POPOUT-LITE does not have this capability. POPOUT-PRO is an example of stereogram software that does allow the user to specify the resolution of the medium on which the stereogram will be printed or displayed. Using this information, the software can correct this distortion. This is done by trading levels for linearity, which means that the software may not necessarily create a level of depth when the depth file instructs it to do so. In effect, it remaps the pixel intensities of the depth image to correct for the error.

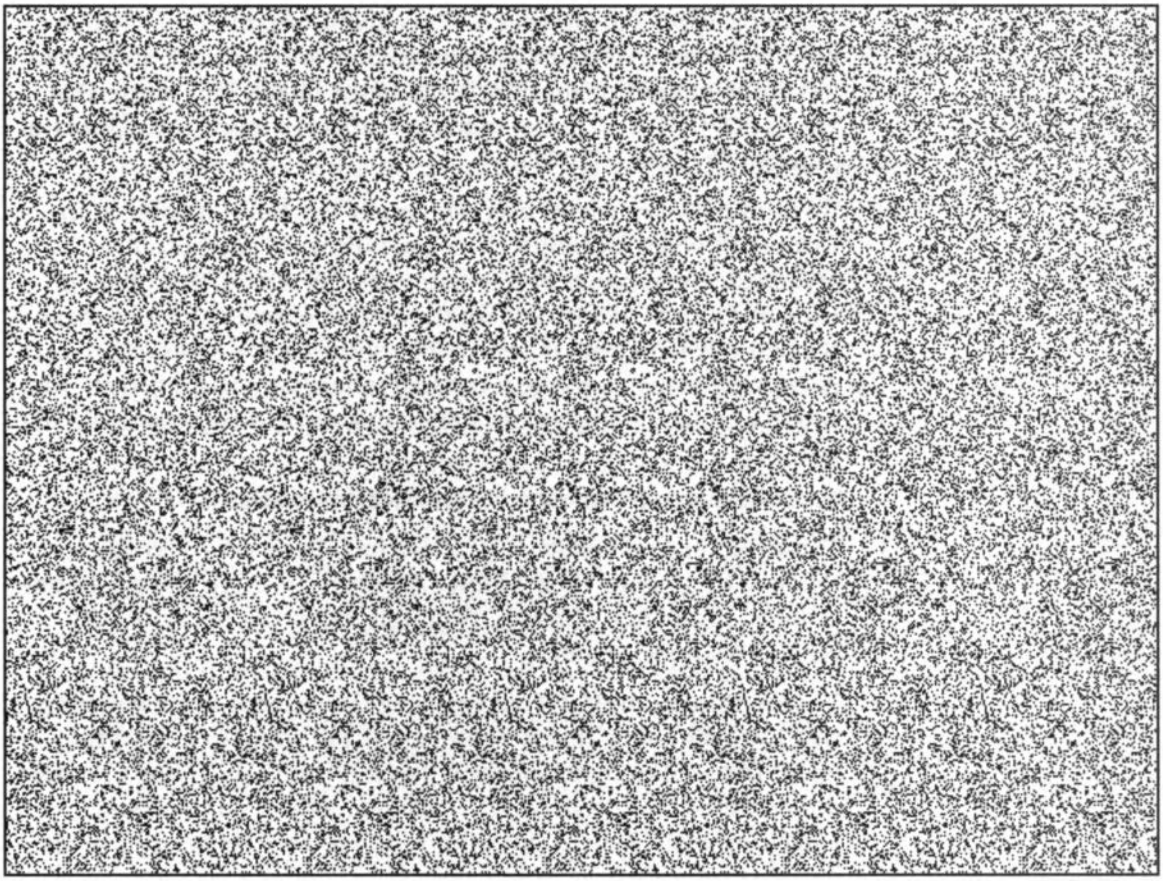


Figure 6.8

Example of depth linearity correction.

See if you can tell the difference between the top and bottom half of the wedge image. In the top half, the sides of the wedge tend to bulge out. In the bottom half, they're flatter, as I had intended them to be. The bulge in the sides of the top wedge follows the curve just like in the graph of figure 6.7, where the sides of the bottom wedge are closer to the straight line in the graph.

The Shift and the Shrink

I'd like for you to be aware of one final, obscure topic. I said earlier that I believe the stereogram generation methods I've presented here are the easiest to understand, and the least math-intensive.

This topic applies to all types of stereograms, regardless of whether or not they've been created with random dots or colorfields, and regardless of the number of apparent levels in the image. This means that it affects POPOUT-LITE just as much as it affects a much more powerful program like POPOUT-PRO.

Up until now, I've passed over and totally ignored a phenomenon that occurs when creating a stereogram using the methods I've presented. For purposes of this discussion, I'll refer to this phenomenon as *the shift*. As with a lot of these more obscure topics, perhaps the best way to explain it is to start by showing you an example. Let's go back for a minute to the text-based stereograms that I first showed you in chapter 2. I'm going to start with a depth image that looks like this:

```
00000000000000000000111111111100000000000000000000
```

This depth image consists of 50 characters. There are 20 zeroes, followed by 10 ones, then another 20 zeroes. In short, I've got 10 ones centered in a bunch of zeroes. From this, I'm going to create a 50-element-wide stereogram.

To keep things simple, I'll use the same ten-element starting pattern that I've been using, that is, 0 through 9. I'm going to process the image from left to right, and treat it as if it were going to be a random dot image, so I'll introduce a new element when I do the insert. Forgetting about pattern substitution and to keep things simple, the resulting stereogram looks like this:

```
01234567890123456789123456789 1A234567891A234567891
```

Do you see how I came up with this line? If not, you may want to review some of the concepts of chapter 3.

Starting at the left, there are ten fuzzy numbers, 0 through 9. They're fuzzy because while my right eye sees them, my left eye sees only white paper. Following these, there are ten clear numbers, also 0 through 9. Then, there is a fuzzy zero that only my left eye sees. This is a discrepancy that's going to produce a left edge of a raised region. Following the fuzzy zero are ten clear numbers that appear to be raised. The ten numbers are `1234567891`. Then, there is a fuzzy `A` that only my right eye sees. This is another discrepancy that causes the right edge of the raised region. Following the `A` are 19 clear characters. They are `234567891A234567891`. Finally, there are ten fuzzy characters that only my left eye sees. These characters are `A234567891`.

Okay, let's total them up. Because there are ten fuzzy characters on both the left and right sides, let's ignore them. Also, because there is a fuzzy character on either side of the raised region, let's ignore them, too.

What's left? There are ten clear characters at level 0, followed by ten clear characters at level 1, then followed by 19 clear characters back at level 0. Notice that it's 19, not 20. See a problem? I've got a ten-character raised region all right, but what happened to it? See how it's not centered, as the depth file specified it to be? The raised region is definitely and obviously shifted to the left of center!

Without getting into a whole lot of detail, I want to show you the phenomenon and present a simple solution to the problem. For the sake of simplicity, let's say that there are 20, not the actual 19, elements to the right of the raised region. This would give us a stereogram with ten elements at level 0, 10 at level 1, and 20 at level 0—a total of 40 clear pixels. Using these round numbers, you can see that our stereogram has shrunk from the original 50-element depth image by ten elements. Furthermore, all the shrinking appeared on the left side of the image, where ten elements disappeared!

A very simple yet effective solution to this problem is to shift the depth image five elements to the right before processing the image. Why five? If I shift the image five elements to the right, I will even out this ten-element shrink and center the raised region. Let's say I were to add five zeroes to the left of the depth image, and remove five from the right. The result looks like this:

```
00000000000000000000000001111111111000000000000000
```

Now, if I create a stereogram just like I did before, the result looks like this:

```
01234567890123456789012346789012346A789012346A7890
```

Notice that the raised region now appears to be centered.

Another option is to add an additional five zeroes to both ends of the depth image, making it a total of 60 elements wide. The result is a stereogram that is wider than the depth image; for some types of images, this not only may be acceptable, but also may be desirable. When viewed divergently, there are 50 clear pixels in the image, just as there were 50 elements in the depth image. In this way, I can adjust for the ten-element shrink.

Some of you may have realized that the amount of shift and shrink is dependent on the pattern width. With an initial pattern width of ten, the image appears to shift ten elements to the left, and shrink by ten elements. I account for the shift by shifting the depth image half the starting pattern width to the right. I account for the shrink by increasing the depth image width by half the pattern width on the left, and half on the right.

Summary

With this chapter, I have given you an introduction to some of the less obvious problems that can arrive in stereogram generation. Some of these problems are specific to the stereogram generation methods presented in this book, but other methods have their own set of problems.

If nothing else, I showed you that even with the basic concepts presented in chapters 2 and 3, you must take into account a number of other aspects having to do with the stereogram-generating software.

The following topics were covered in this chapter:

- Many stereograms that are created do not involve the use of random dots. Technically, they are not RDS images, but are more commonly referred to as colorfield stereograms. These images obtain their substitution pattern not from a random selection of dots, but from a second image called the colorfield image. The result is a stereogram that consists of separate colorfield tiles, where the individual tiles have been distorted to create the hidden image.
- Some special considerations must be made by the stereogram software when creating colorfield stereograms. In particular, the insertion of elements into the current pattern must be done while taking into consideration the width of the colorfield image. Because the substitution pattern is one line of the colorfield, the total number of pattern elements cannot exceed this colorfield line width.

- Common stereograms have been designed so that they are properly viewed when the observer's point of focus is located at a point behind the plane of the image. This is referred to as divergent viewing. This same image can also be viewed in such a way that the observer's point of focus is located in front of the plane of the image. This is called convergent viewing. Viewing a common stereogram in this manner results in the observer seeing a mold of the intended hidden image. Stereograms can just as easily be created with convergent viewing in mind.
- Some interesting effects can be obtained using the concept of filtering, or smoothing. When a depth image is smoothed or blurred before being processed by the stereogram software, many more levels of apparent depth can be created. This technique works especially well with colorfield stereograms.
- Other equally interesting effects can be obtained by creating a stereogram in the normal fashion, and then by using some image processing software to smooth or blur the stereogram.
- A stereogram can be thought of as an example of applied geometry. Because of this, there are some distortions that can arise that follow some simple rules of geometry. Specifically, this chapter has shown you that the apparent distance between two levels is not a constant—it varies as the levels get closer to the observer. The result is a distortion in the apparent depth of the image. This distortion is easily explained, and can be corrected by the stereogram software.
- When creating stereograms using the techniques I've shown, there is a phenomenon that occurs that causes a noticeable, horizontal shift in the image. The direction of the shift is dependent on the direction that the line is processed. This problem is easily corrected, and is done automatically by the stereogram software.
- Because viewing a stereogram involves your two eyes looking at two different points on the image simultaneously, the left and right edges of the stereogram appear to be fuzzy. This is important to realize when creating depth images because any detail at the edges of the depth image may not be visible in the final stereogram.

Part II: Generating Random Dot Stereograms

7

Creating a Depth Image with Paintbrush

The previous chapters introduced the concepts behind the development of random dot stereograms. If you're a programmer, the wealth of information presented will give you a good starting point in order to develop your own RDS generation programs. On the other hand, if you just want to have fun creating RDSs, you are ready to start generating them. The first step is to use a paint program to create a drawing that will be read by POPOUT-LITE—the RDS generation program included with this book.

Why Use Paintbrush?

In order to create drawings to convert into stereograms, we need to use a paint program. Lots of commercial and shareware programs are out there that will do the job. In this book, we use Paintbrush to develop such drawings for the following reasons:

- Paintbrush is included with each copy of Windows; it's accessible to most computer users.
- Paintbrush has a good set of tools that will let you create .BMP-format drawings that can be read by POPOUT-LITE. The tools are simple to use and they let you concentrate on creating drawings rather than on finding out how they work.
- POPOUT-LITE is also a Windows application; it makes sense to have both applications in the same operating system.
- By using Windows, you can run both POPOUT-LITE and Paintbrush and switch between the two easily.

Once you learn how to develop drawings, you can switch to more complex paint programs if you wish. At this point, you just need to learn what effects can be created by means of the objects you create and the colors you select.

Let's present briefly some of features available in Paintbrush. As we move along, we'll try to cover most of the program.

TIP: You can always get additional help by clicking the Help command or by pressing F1 inside Paintbrush.

Working with Paintbrush

In order to run Paintbrush, run Windows and double-click the Accessories Group icon. After double-clicking, you should see something similar to figure 7.1.

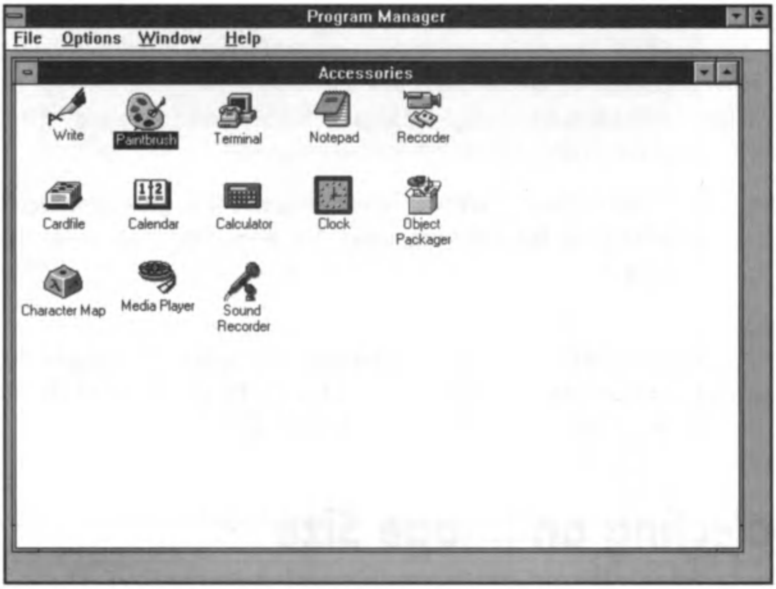


Figure 7.1

The Accessories group inside Windows.

Now, double-click the Paintbrush icon. This executes the program and you should see something similar to figure 7.2.

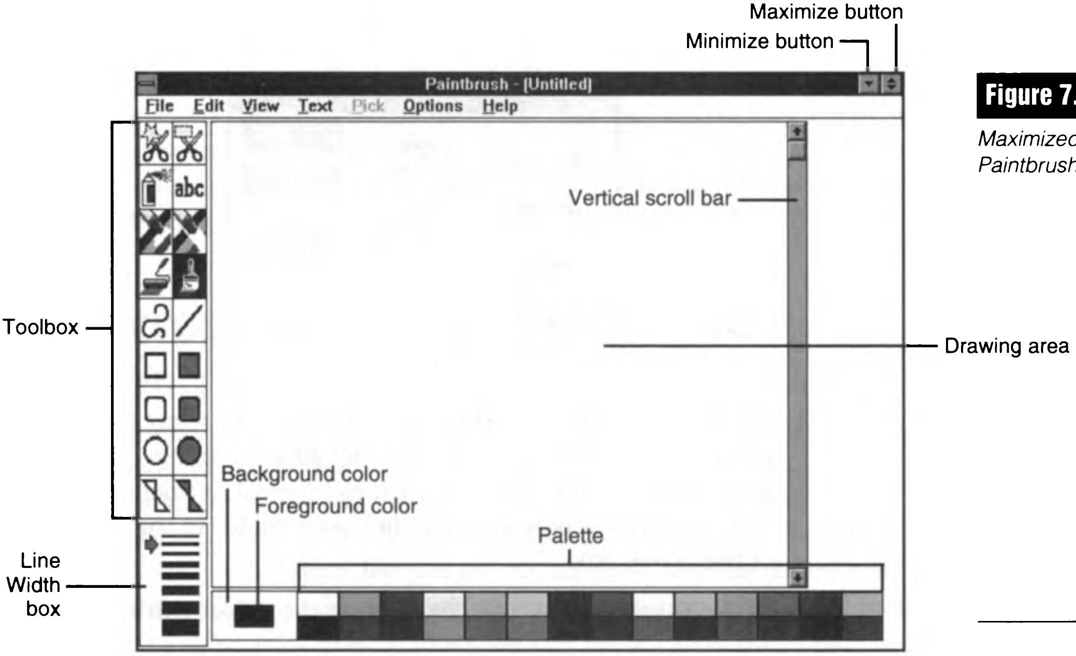


Figure 7.2

Maximized view of Paintbrush.

NOTE: Figure 7.2 presents a maximized view of Paintbrush. To maximize a window, click the icon that appears at the upper right corner of that window. This should resize it to fill the screen completely.

When drawing with a paint program, it is good practice to maximize the drawing area. This gives you maximum drawing area.

Once Paintbrush has been loaded, you're ready to create drawings. Let's now take a look at how to select the final size of the drawing and how to view the drawing as we develop it.

Selecting an Image Size

Paintbrush can create drawings with different sizes. The *size* of a drawing basically refers to its resolution (640 by 480 pixels), or to its dimension (inches). To select the size of the drawing:

1. Click the **O**ptions menu in the menu bar and click **I**mage Attributes to display the Image Attributes dialog box (see fig. 7.3).

Figure 7.3

Image Attributes dialog box.

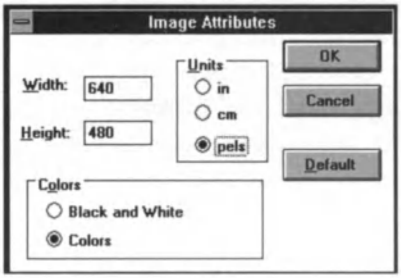


Image Attributes

Width: 640

Height: 480

Units: ○ in ○ cm ● pels

Colors: ○ Black and White ● Colors

2. The Image Attributes dialog box enables you to select different parameters for the image that you are about to create. First, the size of the drawing can be defined in terms of in (inches), cm (centimeters), or pels (pixels). Click your choice in the **U**nits area of the dialog box.

3. In the **H**eight and **W**idth boxes, enter the values you choose.

NOTE: For the sake of simplicity, we use pixels as our standard dimension unit for all drawings. Unless otherwise indicated, we use 640 as the **W**idth and 480 as the **H**eight.

4. Another parameter, C**o**lors, is used to indicate whether you want to create drawings in black and white or in color. Choose a C**o**lors option by clicking it.

NOTE: Through the book, we use mostly color drawings, because these enable us to create stereograms with multiple depths.

5. To exit the Image Attributes dialog box, click the OK button.

Scrolling through a Drawing

Most drawings that you'll develop in Paintbrush will be larger than the window where they are displayed. In order to view other areas of the drawing, use the horizontal and vertical scroll bars.

Where you click in the scroll bar determines how large a scroll will be. Clicking any of the arrow buttons at the end of a scroll bar generates small displacements of the drawing, but clicking anywhere in-between these buttons generates larger displacements. A faster method is to drag the scroll box (the square box in the scroll bar), which moves the drawing by a proportional amount.

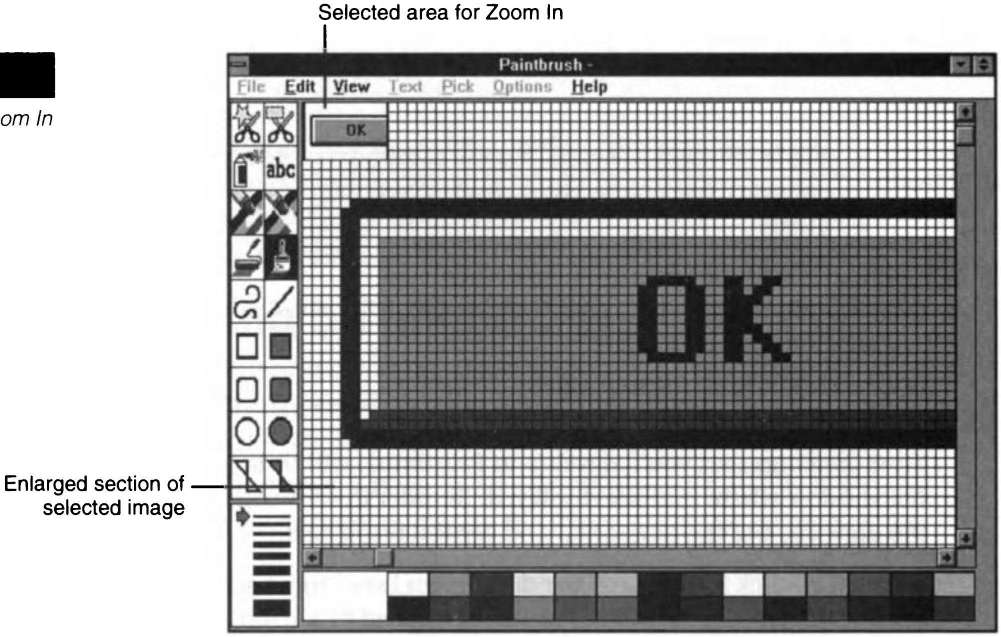
Full Screen and Detail Viewing of a Drawing

Paintbrush lets you see the whole drawing. To do this, select the **V**iew menu and click **V**iew Picture (Ctrl+P). The image is resized to fit the whole screen. To return to the previous view, click anywhere on the drawing. The **V**iew menu offers another command called Zoom **O**ut (Ctrl+O). This feature lets you see the full drawing inside the current window. Because the whole object is displayed in such a reduced area, you might find it difficult to see some of the details in the image.

An additional command on the **V**iew menu is Zoom **I**n (Ctrl+N). This command lets you select an area of the drawing to enlarge. Once in this view, you can edit individual pixels by clicking them. In this view, all pixels look like square, colored boxes (see fig. 7.4). To return to the previous view, select **V**iew and click Zoom **O**ut (Ctrl+O).

Figure 7.4

Effects of Zoom In command.

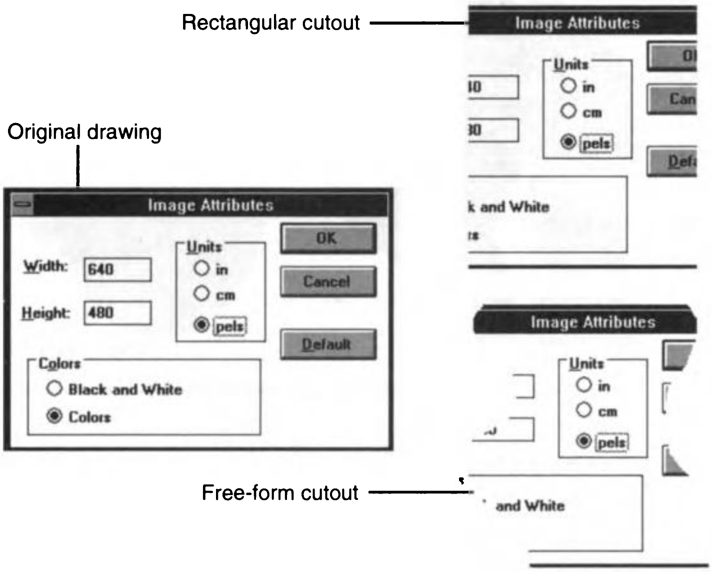


The Paintbrush Tools

To draw or modify anything on-screen, you need to select a tool from the Toolbox. The Toolbox consists of 18 tools that you can select to perform everything from drawing lines, rectangles, and ellipses to adding text and selecting areas on-screen. I'll discuss each one briefly and give examples where appropriate.

Cutout Tools: Scissors and Pick

The Cutout tools are on the top row of the Toolbox. *Cutout* refers to an area on the drawing which has been selected with one of the Cutout tools: Scissors and Pick. The Scissors tool (which looks like scissors with a dotted-line star shape) can select nonrectangular areas on-screen. These areas are referred to as *free-form regions*. The Pick tool selects only rectangular regions from a drawing. Once these areas or regions are selected, you can cut, copy, or move them as required. Figure 7.5 presents examples of free-form and rectangular regions. To compare both, see figure 7.5. The original drawing (the Image Attributes dialog box) is on the left side of the image. On the right, you'll see two cutout regions. The one on the upper right was taken from a rectangular cutout region, while the other on the lower right was taken from a free-form cutout region. You can almost follow the curved contour of the cutout.



Width: 640
Height: 480
Units
○ in
○ cm
● pels
Colors
○ Black and White
● Colors

Figure 7.5

Rectangular and free-form cutout regions.

Once a cutout has been defined, you can apply one of the following operations: cut, copy, move, or shrink/enlarge. If you choose cut or copy, you can also select the paste operation. Let's discuss each operation.

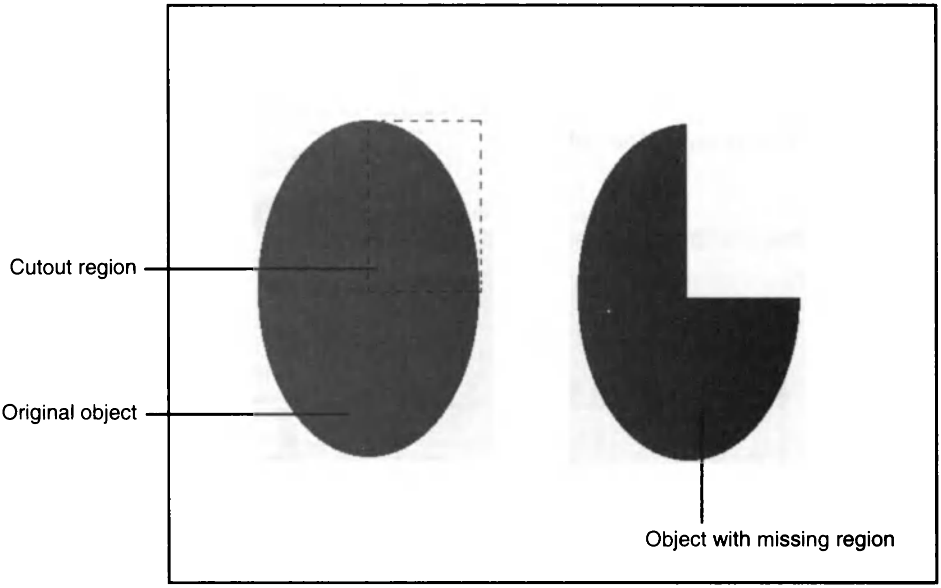
Cutting and Pasting a Section of the Drawing

Figure 7.6 presents an original object and the cutout region selected. To its right, the result of a Cut operation is shown. To perform a Cut operation:

1. Select either the Scissors or the Pick tool and define the cutout region in the drawing.
2. Next, from the **E**dit menu, select the Cu**t** (Ctrl+X) command. This erases the cutout region.

Figure 7.6

Cutting a rectangular section of an object.



3. The part of the drawing that has been cut is not lost. At this point it's saved into the Clipboard and can be retrieved by Paintbrush or other programs. To place this cutout in another part of the drawing, click the **E**dit menu and select **P**aste (Ctrl+V). The paste operation places a copy of the cutout in the upper left-hand corner of the drawing.
4. To move the cutout to its new position, simply move the mouse until it's positioned on top of the cutout. Now, press and hold the left mouse button and move the mouse to the new location. The cutout follows the mouse as long as you have the button pressed. To drop the cutout in its new location, release the left mouse button.

If this operation sounds a little drastic, you can use the **C**opy (Ctrl+C) operation to replicate the cutout region somewhere else in the drawing.

Copying and Pasting a Cutout

The copy/paste operation is nearly identical to the cut/paste operation. The difference is that the copy operation simply places a copy of the cutout region inside the Clipboard and it doesn't erase that area from the place it was copied from. Figure 7.7 presents an object of which a part has been copied/pasted several times.

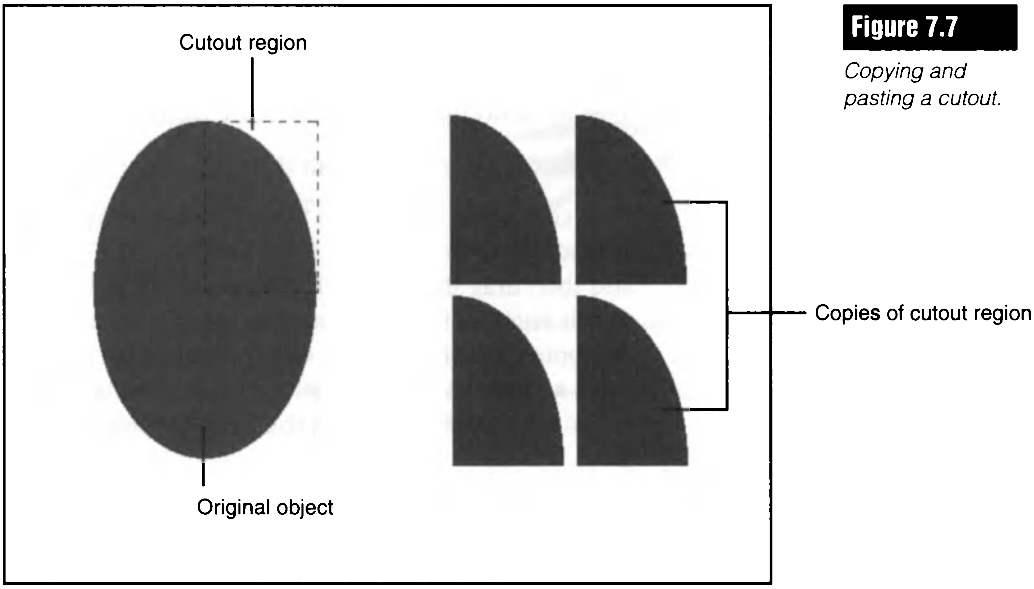


Figure 7.7

Copying and pasting a cutout.

The sequence is simple: select the cutout region and then select **E**dit **C**opy (Ctrl+C) operation. Select **E**dit **P**aste (Ctrl+V) and position the cutout as required.

Moving a Cutout

When you create a shape or drawing and you want to move part or the whole drawing to a new location, use the Move operation:

1. Select the cutout region with one of the Cutout tools.
2. Now, move the mouse to a location inside the cutout region and press the left mouse button. This tells Paintbrush to make the cutout a *floating selection,* which can be moved anywhere as long as the button is pressed.

3. Hold down the left mouse button and drag the cutout to its new location. To release and drop the cutout on its new location, release the left mouse button.

Shrinking and Enlarging Cutouts

Paintbrush lets you select a cutout from a drawing and resize it as needed. You can scale a cutout keeping the same proportion (also called *aspect ratio*) as the original cutout or simply scale it to any proportion. To resize a cutout:

1. Start by defining a cutout from any region in the drawing.
2. Choose **S**hrink + Grow from the **P**ick menu.
3. Now comes the tricky part: Select the region where the new resized cutout will appear. Click and hold the left mouse button, and then drag the mouse to define a rectangular region. The size of this region will determine the region of the final object. At this point, the proportions of the final object are defined by the box's aspect ratio. If you wish to maintain the same proportions as the original cutout, press the Shift key as you drag the mouse.
4. To finish sizing the object, release the mouse button.

Figure 7.8 presents various examples of scaling an object. The original object is a lightning bolt. To show the region defined, I created a black background to the right of the original object. Once the new object is placed there, the white background of the bolt will define the scaling box used (I explain how to work with boxes in later sections). The first row of copies were created maintaining the original proportion. The rest were defined based on whatever fit into the area.

Airbrush Tool

The airbrush tool, which looks like a spray can, is a tool that's not really useful in the creation of images for RDS generation, but I'm sure that some of you will find a way to make me eat my words! Basically the airbrush simulates—yes, you guessed it!—an airbrush effect. Figure 7.9 presents the effects of moving the mouse slowly or quickly when using the airbrush tool.

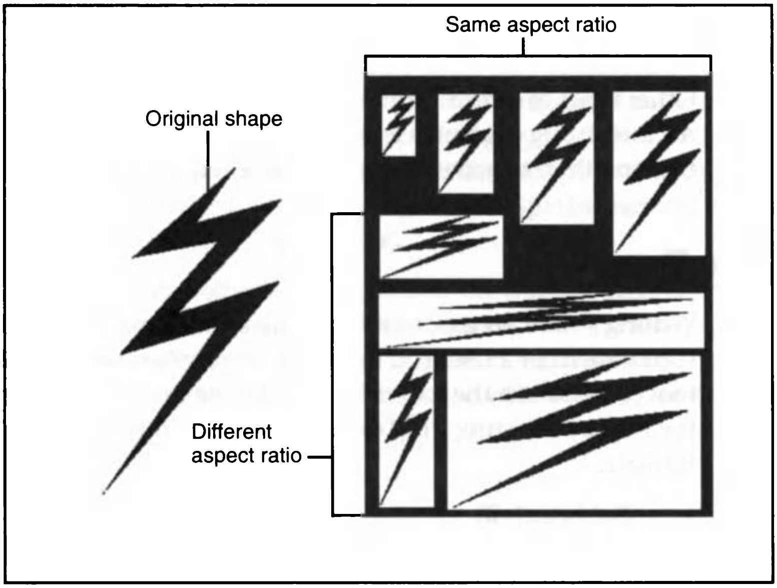


Figure 7.8

Copying a cutout with different scale factors.

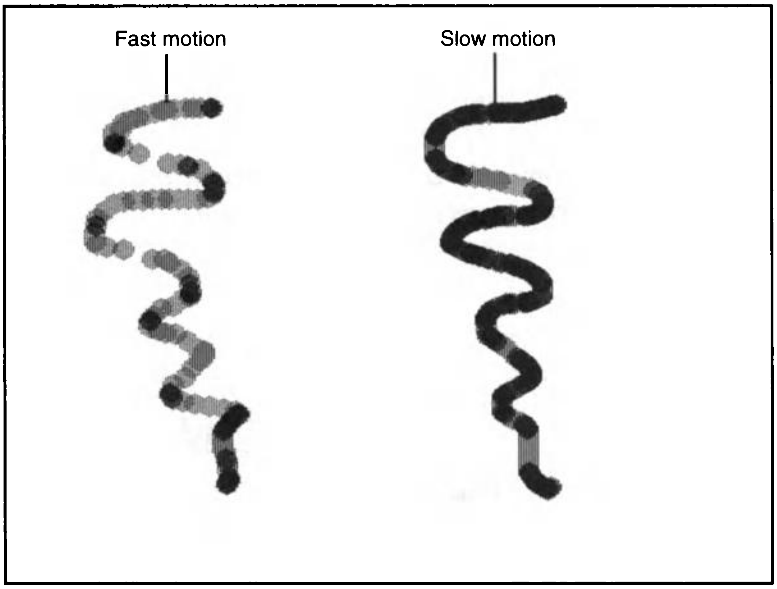


Figure 7.9

Effect of moving the airbrush tool slow and fast.

To draw with the airbrush tool, select it and move the mouse to the drawing area. To start drawing with it, press the left mouse button as you move the mouse. Remember that the motion and speed of the mouse determine the final marks on the drawing area.

Other types of airbrush shapes are available under the **O**ptions menu. An interesting challenge is to use the airbrush to create the leaves of a tree and then to generate an RDS. Give it a try if you're up to it!

Text Tool

Writing your own name and creating an RDS from it is almost a requirement or a law. You can write text by first selecting the Text tool (it looks like the letters "abc." Change the style and size of the font by selecting the **T**ext menu. There you find the following features:

Bold (Ctrl+B)

Italic (Ctrl+I)

Underline (Ctrl+U)

Outline

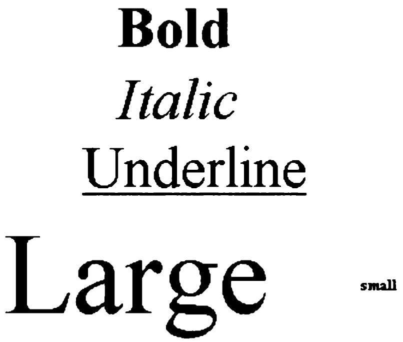
Shadow

Fonts...

The **F**onts command enables you to change the style and size of the fonts you wish to use.

Figure 7.10

Different font styles and sizes.



Bold

Italic

Underline

Large small

After you've selected all the attributes you want, simply click in the drawing area and type the text you want.

Roller and Brush Tools

These tools look just like the painting tools they're named for. The roller basically fills any closed area with the foreground color (refer to the section "Selecting Foreground and Background Colors" for more info on how to select these). Be sure that the area is closed. If there is a gap in the area, the fill color will bleed through the gap and eventually cover a lot more of the drawing.

In figure 7.11, the roller is applied inside the small square. In the closed area case, only the small square is filled with the foreground color. In the other case, as the area is filled, the gap permits the procedure to fill the outermost closed area, which is the larger square.

The roller works by flooding a closed area in the screen with the foreground color. This means that the color follows a behavior similar to a liquid that is poured into a container. If the container has a small hole, the liquid will also come out through that hole. That is what happens when an area that appears to be closed, such as a polygon, starts bleeding the color.

To use the roller, select it from the Tools area and click the left mouse button on the closed area you wish to fill with the foreground color. If the color bleeds, select **U**ndo (Ctrl+Z) from the **E**dit menu. This backs up a step. Use the Zoom **I**n (Ctrl+N) to find out where the area is open.

The brush is a handy tool because it lets you draw in a sort of freeform fashion. By moving the mouse around the screen you can create wavy curves and complex shapes (see fig. 17.12).

To draw with the brush, select it from the Tool area and move the mouse to the drawing area. When you wish to start drawing, simply press the left mouse button and move the mouse. To stop drawing, release the mouse button.

If you wish to modify the brush, select the **O**ptions menu. Once there, select **B**rush Shapes. Make your change, and then click OK.

Lines and Curves

Lines are one of the easiest tools to use. Simply click the mouse to create one end of the line, and then click again to place the other end.

Use the Line Width box at the bottom of the tool palette to change the width of the line to be created.

Figure 7.11

Roller tool example: Closed area and area with a gap.

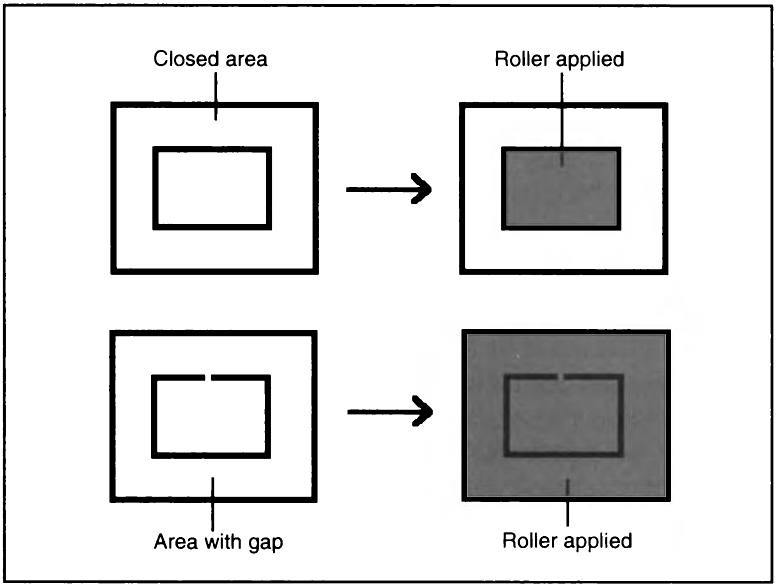
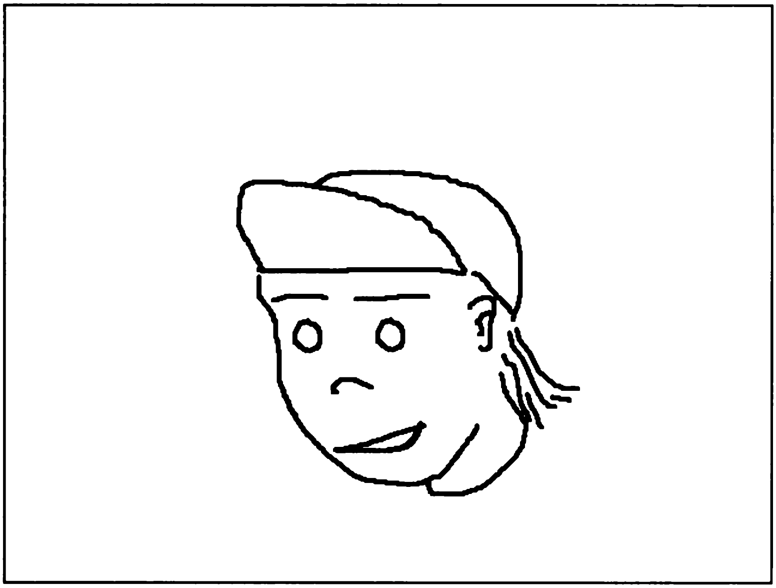


Figure 7.12

Using the brush tool for free-form drawing.



The Curve tool can create a curve based on four points. The first two points are the ends of a straight line. The other set of two points basically attracts the line and bends it. Figure 7.13 illustrates the following steps:

1. Select the Curve tool. To create a line, click the left mouse button on the screen and drag to the point at which you want to end the line. Release the button. You have a straight line.
2. Create a point near one end of the line by clicking the left mouse button. Observe what happens to the line: it becomes a curve—almost as if it is attracted to the point you created.
3. Create another point near the other end of the line and watch the line curve toward it.

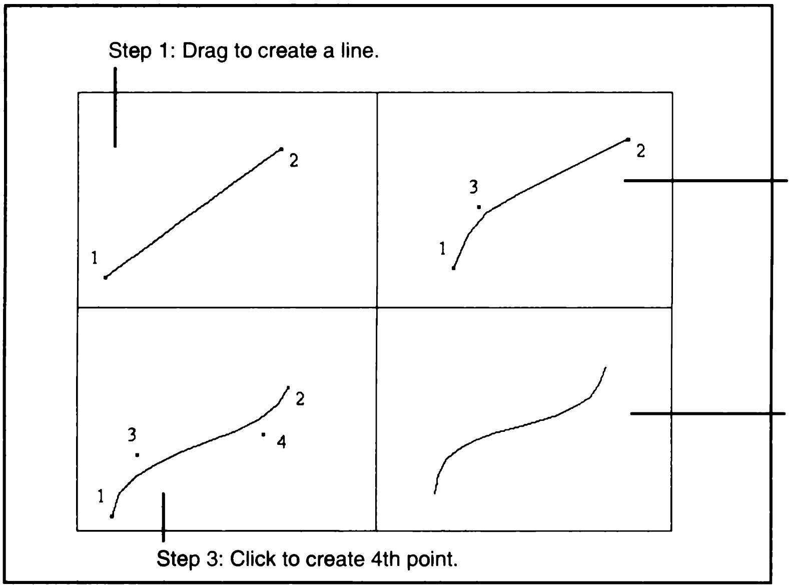

Step 1: Drag to create a line.
1
2
3
1
2
3
4
1
2
Step 3: Click to create 4th point.
Step 2: Click to create 3rd point.
Step 4: Done!

Figure 7.13

The Curve tool step-by-step.

A second way to create curves is to draw a straight line using the Curve tool, click a point on the line, and drag. Drop the curve where you want it by releasing the mouse button. To adjust the curve, click a point on the curve and drag. Release the mouse button when you have the curve you want.

Box Tools

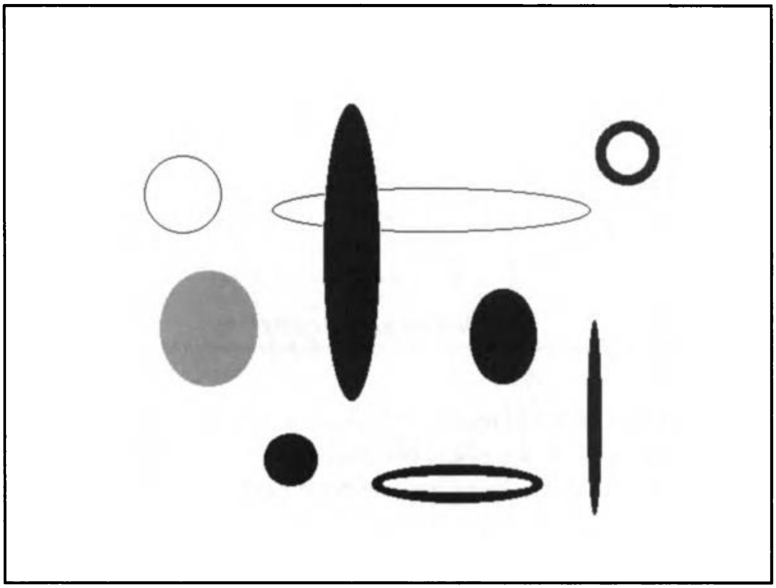
Four box tools let you create borders, filled-in boxes, and boxes with square corners or rounded corners. Before you create a box, you can use the Line Width box to change the line thickness of the box itself. To create a box, first click the appropriate box tool, and then click the left mouse button anywhere on-screen. This point defines one of the corners of the box. Hold down the left mouse button and move the mouse where the other corner of the box will be. Release the mouse button.

Circle Tools

The Circle tools also let you create ellipses. Similar to boxes, you can either create the outline only or fill them with the foreground color. Both objects are created by clicking the appropriate circle tool, and then by dragging to define a boundary box. If the box is square (which you make by holding down Shift as you drag the mouse), the object will be a circle. Creating a rectangular box creates an ellipse. Figure 7.14 shows various circles and ellipses.

Figure 7.14

Circles and ellipses.



Polygon Tools

Polygons consist of a set of points interconnected by lines. Similar to boxes and circles, they can be drawn just as outlines or as filled objects. To create a polygon, select the polygon tool, click the left mouse button on-screen, hold the mouse button and drag to where you want the first line to end. Release the mouse button. Now move the mouse to where you want the second line to end and click the left mouse button. The second line is formed. Continue to click the mouse where you want the corners to be and you will soon have a polygon. As polygons are closed regions, the last point created should be near the first point created. If the last point is too far from the first one, double-click the left mouse button and Paintbrush will close the polygon for you. Figure 7.15 displays various polygons.

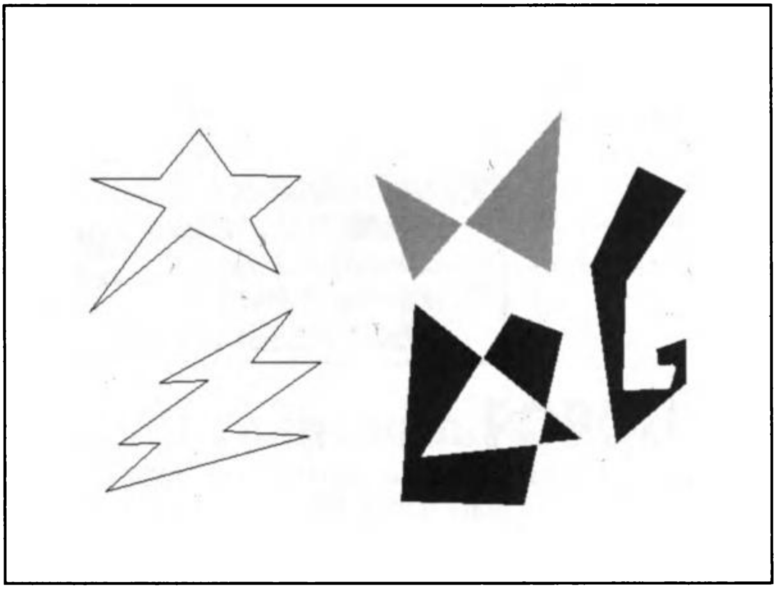


Figure 7.15

Various polygons.

Selecting and Creating Colors

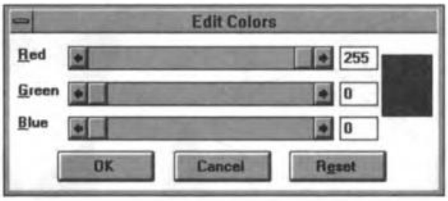
Up to this point, we have dealt with tools that let us create or modify the drawing itself. An important part of this process is the ability to select different colors. The set of available colors, which is referred to as a *palette of colors,* is one of the most important selections when dealing with RDS creation.

The following sections describe how colors can be selected from the Paintbrush palette, how you can modify that palette, and the importance of saving these colors as a set or as a palette.

Selecting Foreground and Background Colors

Paintbrush displays the current palette as two rows of colored boxes at the bottom of the screen (see fig. 7.2). Observe the two rectangles, one inside of the other, that display the currently selected foreground and background colors. To select the foreground color, click the left mouse button on the color that you want. If you wish to select the background color, click the right mouse button on the color which will become the background.

What about modifying any of these colors? To modify a color click the **O**ptions menu and select **E**dit Colors. A dialog box similar to figure 7.16 appears.



Red 255

Green 0

Blue 0

Figure 7.16

Edit Colors dialog box.

To edit a specific color from the palette, simply click the corresponding colored box in the palette. This color will appear in the box displayed in the Edit Color dialog box. Each color is represented by a set of three numbers (from 0 to 255) that describe the Red, Green, and Blue content. For example, the color Green is [0,255,0] and Blue is [0,0,255].

As you'll discover later, the colors specified in the palette are used by the RDS program to determine the drawing's depth information.

Creating Custom Colors: The Palette

Eventually you'll want to experiment with different color palettes. For example, you can easily create a grayscale palette by using colors that have the same value in all three components. For example, you can create a grayscale palette composed of [0,0,0], [50,50,50], [100,100,100] and so on until you end up with [255,255,255].

Once you have created your new set of colors, you can save it to a .PAL file. To do this, select the Options menu and click **S**ave Colors. Define a good name for it, for example GRAY.PAL for your grayscale palette.

Loading Other Palettes

Why is all discussion about palettes important? POPOUT-LITE includes a predefined palette which must be used to create your drawing. POPOUT-LITE recognizes these colors and has already defined what the relative depths are between them. To load the POPOUT-LITE palette, select the **O**ptions menu and click **G**et Colors. Change to the drive and directory where POPOUT-LITE resides (C:\POPLITE); select the file POPOUT.PAL by double-clicking it.

Observe that as soon as the palette is loaded, the palette in Paintbrush is updated. The next sections describe this in detail.

Using Paintbrush with POPOUT

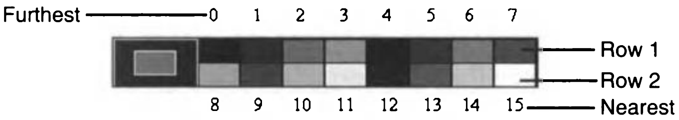
The whole idea of learning how to use Paintbrush is to be able to use it to create drawings that eventually will become random dot stereograms. The next sections discuss the key elements behind using the POPOUT.PAL palette, and give you instructions for drawing an image that you can process with POPOUT-LITE to create an RDS.

The POPOUT Palette

Before you do any drawings in Paintbrush that you want to process with POPOUT-LITE, you must load the predefined palette POPOUT.PAL (see fig. 7.17).

Figure 7.17

POPOUT-LITE palette.



How do you determine the relative depths between the colors? Figure 7.17 shows the boxes labeled with numbers, starting with 0 and ending with 15. The lower the number, the further it is from the observer. For example, color 5 is further from the observer than color 7.

In other words, let's assume that we have two circles and one of them is further away than the other. The furthest one should be color 5 while the nearest one to the observer should be color 7. Use color 0 as the background color. This guarantees that it will be the furthest of all objects in the drawing.

Creating a Simple Drawing

This section explains how to create a drawing similar to figure 7.18.

Start by loading POPOUT.PAL (you might find it necessary to change to the drive and directory where POPOUT-LITE resides). Now, from the **F**ile menu, select **N**ew. By doing this, you make the current palette active. Observe how the background color now reflects that of the new palette.

Because our background is already using color 0 (the furthest from the observer), select color 1 for the frame and the letters. For the frame, select the box with rounded corners and create a filled-in box with rounded corners. To create the hole in the box, simply select color 0 as the foreground color and create a smaller box with rounded corners. After creating the final frame, select color 1 as the foreground color.

To add text to the image, select the font "ZapfHumnst BT," size 72 and **B**old style from the **T**ext menu, **F**onts submenu. Write your name, that should make the image more interesting. Now, select the Text tool and write whatever text you have chosen. Since I chose the letters "RDS," I typed them in and then selected them as a cutout. I did this in order to center the text inside the frame.

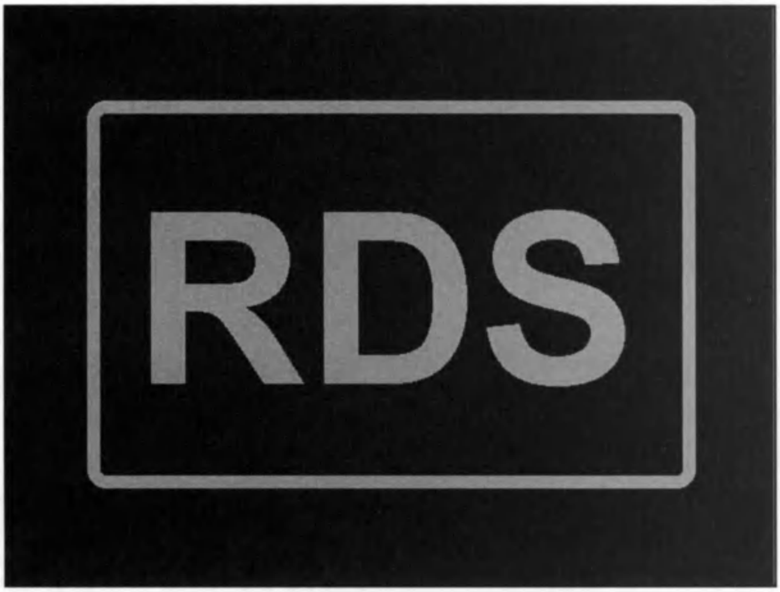


Figure 7.18

A simple drawing used to create the RDS in figure 7.19.

If you observe figure 7.18, the letters are bigger than size 72. I used the enlarge feature inside the Pick menu. I started by selecting the text as a cutout, and then clicked the **P**ick menu. I chose the **S**hrink + Grow operation and enlarged it without maintaining the original proportion.

The next step is important: from the **F**ile menu select Save **A**s and save the image as a "16 Color Bitmap (*.BMP)." This is necessary because POPOUT-LITE only reads 16-color bitmap files. The registered version POPOUT-PRO can read other formats and use up to 256 colors.

Once the file is processed by POPOUT-LITE, your figure should look similar to figure 7.19. You should see a frame and the letters "RDS" (or your name) in the same level. This can't be any simpler! But now that we have done the first image, we should think about how to improve it. The next chapter describes some techniques that you can apply in order to make your final stereograms more interesting.

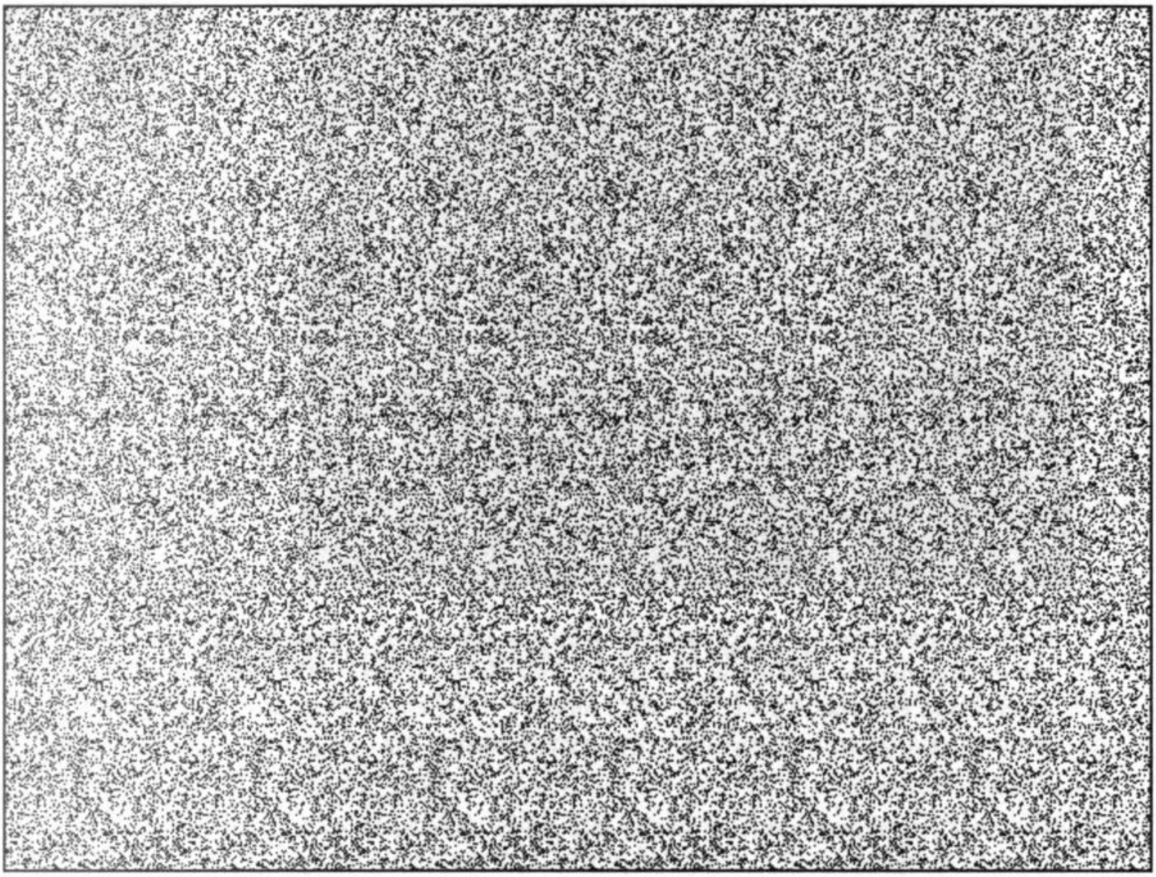


Figure 7.19

An RDS generated using figure 7.18.

Summary

In this chapter, you learned:

- How to use some of the important features behind Paintbrush.
- How to use the predefined palette supplied by POPOUT-LITE.
- How the palette should be used in order to create different levels in a drawing.

8

Drawing Complex Objects

This chapter covers some techniques used to develop complex objects with Paintbrush. If you can't wait to start using POPOUT-LITE, jump quickly to chapter 9 and then return. There are lots of interesting things to cover!

For starters, let's discuss the objects created in the previous chapter (see fig. 7.18 and 7.19). You created a frame and inside that frame you placed the letters "RDS," your name, or some other text. The text basically had two levels: the back level was defined by color 0; the frame and the letters were in the second level, nearer to the observer, defined by color 1.

This chapter explores in more detail how to create other effects such as curved surfaces. In the following sections, you learn to do that and more.

NOTE: Before you do any drawings in Paintbrush that will eventually become RDS images, you should remember the following points:

1. You must load POPOUT.PAL first.
2. The background color should always be color 0. This prevents the background from being closer than objects. you draw. Select color 0 before choosing **F**ile **N**ew.
3. Save the drawing as a 16 color BMP file.

Refer to the section "Using Paintbrush with POPOUT" in the previous chapter for additional information.

Beyond Flat Objects

If you wish to remain interested in random dot stereograms, you must be able to create more interesting effects than just plain, flat objects. Let's take a look at how to create other surface effects.

Making Slanted Surfaces

To create a slanted surface, start by creating a stack of rectangles such as the one shown in figure 8.1. Use color 15 as the foreground color and color 0 as the background color.

Because color 15 in POPOUT.PAL will appear the nearest to the observer, fill each rectangle from the bottom up with each color starting with color 15. In other words, fill the bottom rectangle with color 15, the next one with color 14, and so on. The updated drawing should be similar to figure 8.2.

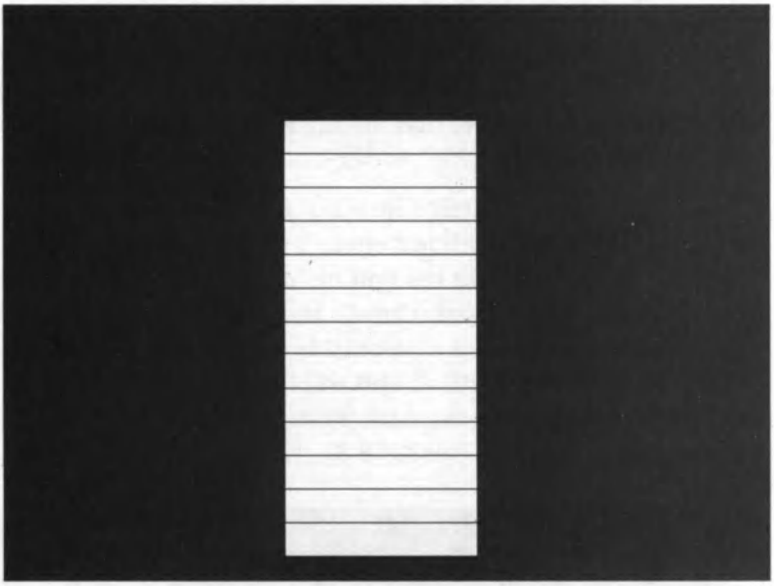


Figure 8.1

Stack of blank rectangles.

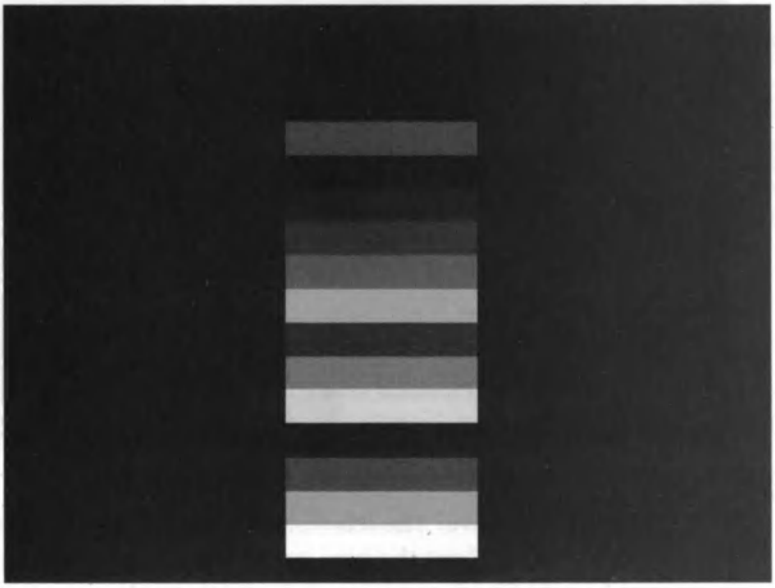


Figure 8.2

Stack of colored rectangles.

Let's stop for a second and evaluate what you've done. First, you created a set of rectangles. Then you filled each rectangle with a color, starting with the lowest rectangle (color 15) and continuing sequentially upward. Remember that the higher the rectangle, the farther it is from the observer.

Now to complicate matters a little bit, select the whole stack as a cutout. Move it to the left just enough to leave room for a second stack of rectangles. Select the **E**dit menu and choose **C**opy. Now from the **E**dit menu select **P**aste (Ctrl+V) and place the stack to the right of the original stack (do not click outside it yet!). Select the **P**ick menu, and then select Flip **V**ertical. This will flip the pasted stack so that color 15 is at the top of the stack. Release the cutout by clicking anywhere in the screen (see fig. 8.3).

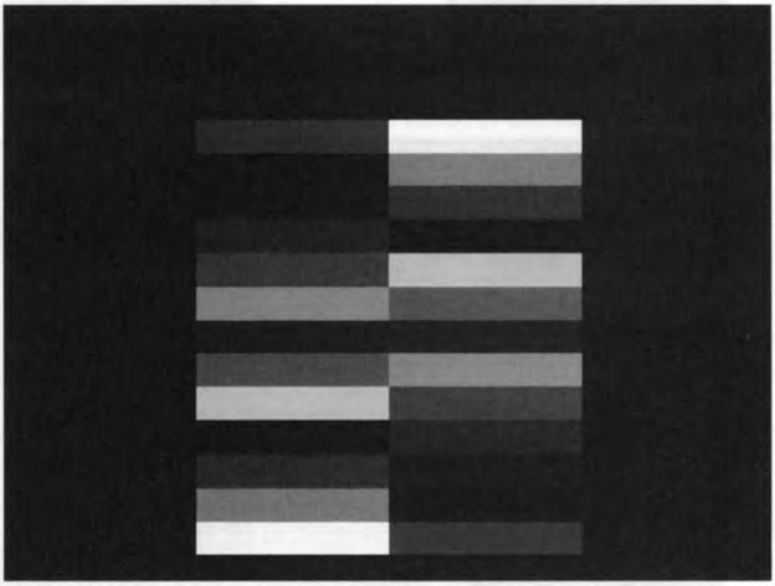


Figure 8.3

Two stacks of colored rectangles.

What effect will this have? The left stack has the bottom part nearest to the observer; what about the new stack? The top part will now be the nearest to you! Check out figure 8.4, which displays the final RDS.

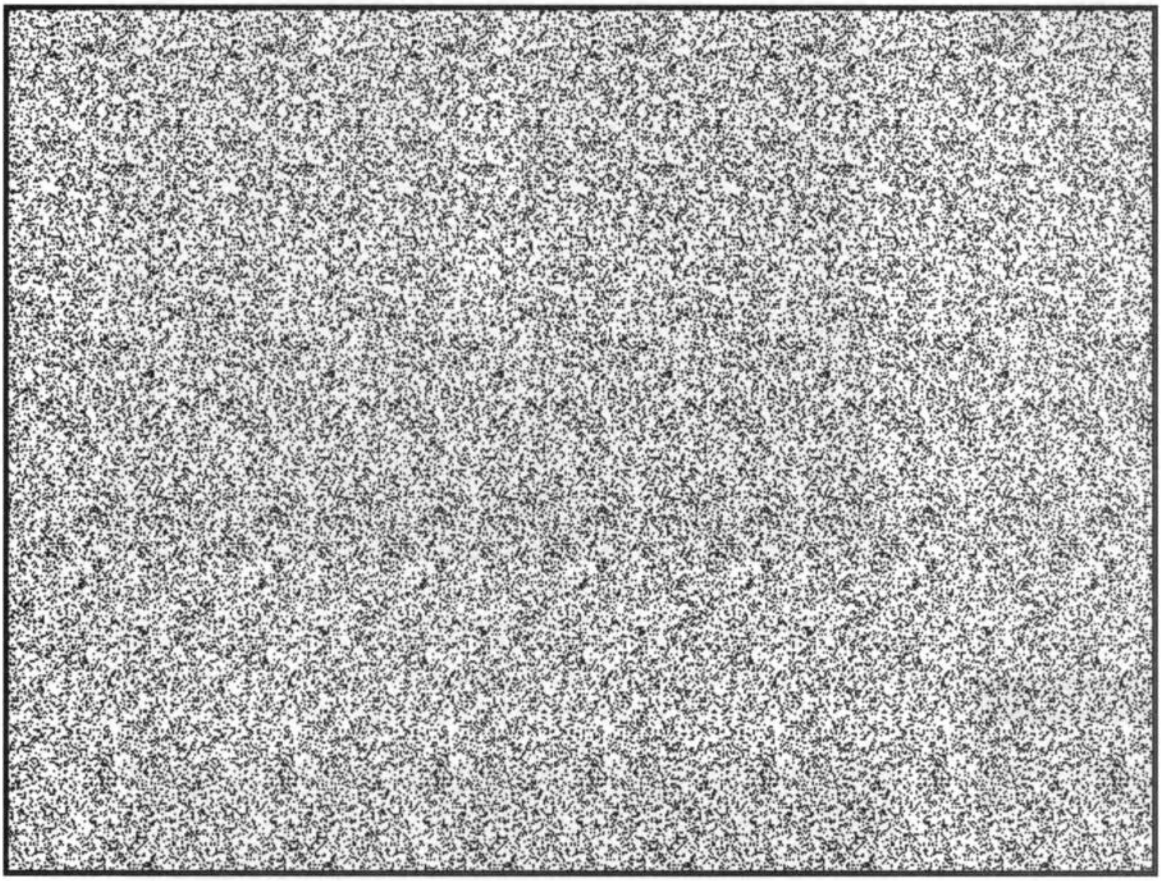


Figure 8.4

Two surfaces slanted in opposite directions.

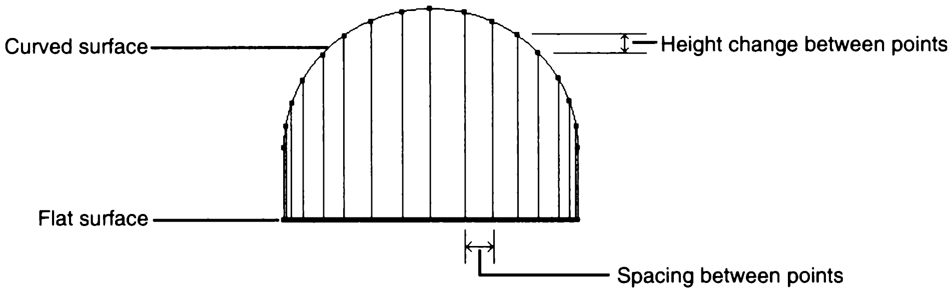
Curving Surfaces

While slanted surfaces can be easily created by painting stacked regions with increasing or decreasing colors, curved surfaces require some additional work. Let's see how you can make a curved surface.

Figure 8.5 displays a half circle that has been subdivided equally.

Figure 8.5

Analyzing curved surfaces.

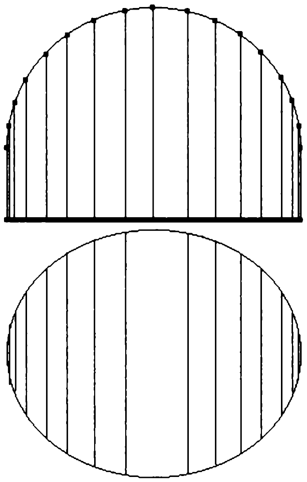


A flat surface is drawn as a thick line and placed below the curved surface. Each point in the curved surface is projected into the flat surface. Observe that the nearer the points are to the top of the curve, the greater the spacing between them. As the points get nearer to the sides of the curve, the spacing between the points diminishes.

You can look at this in another way: the nearer the points are to the top, the smaller the difference in heights between them. Figure 8.6 presents how you can simulate curved surface effects on a flat surface.

Figure 8.6

Applying a curved surface effect on a disc.



To simulate the curved effect, I copied the same distances between the points into the flat surface (in this case, an ellipse). In figure 8.7, I copied the object by itself and filled it with colors. Figure 8.9 is the resulting RDS.

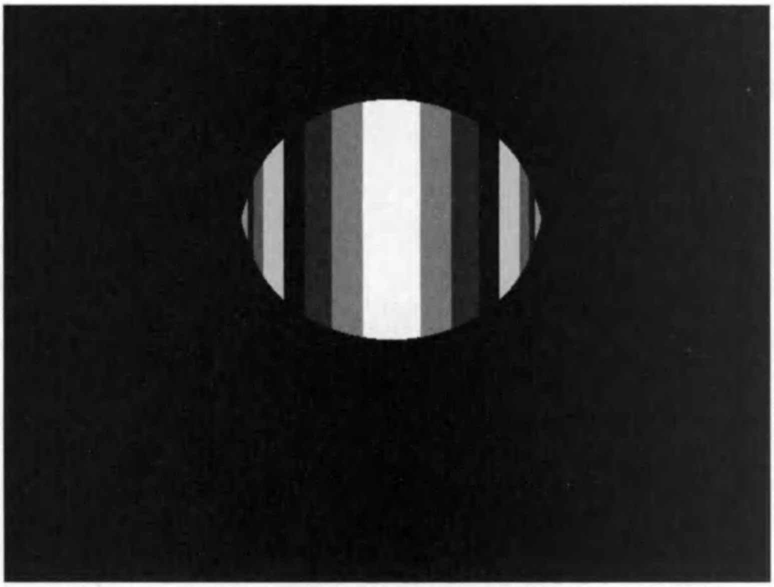


Figure 8.7

Applying a curved surface effect with colors on a disc.

Figure 8.9 appears to be curved horizontally, that is, from left to right. To create a round object, you need to use circles or ellipses rather than just plain polygons. Figures 8.8 and 8.10 display the concept.

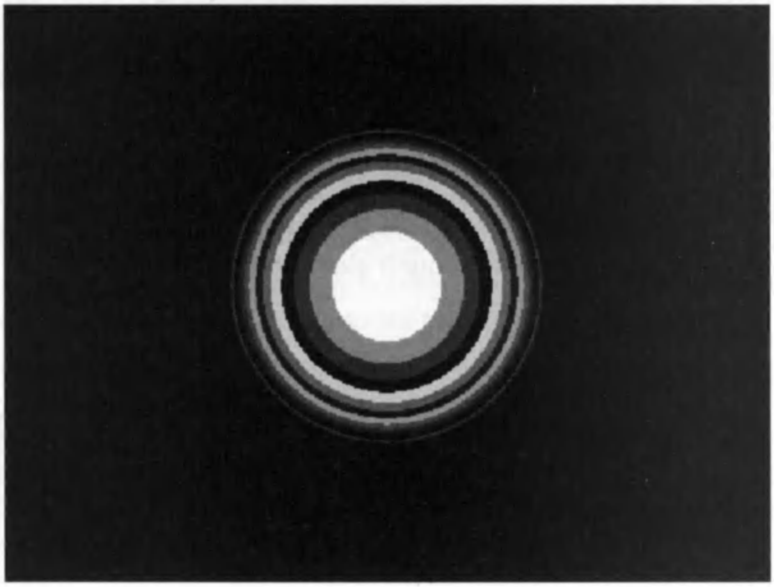


Figure 8.8

Forcing a flat surface to appear round.

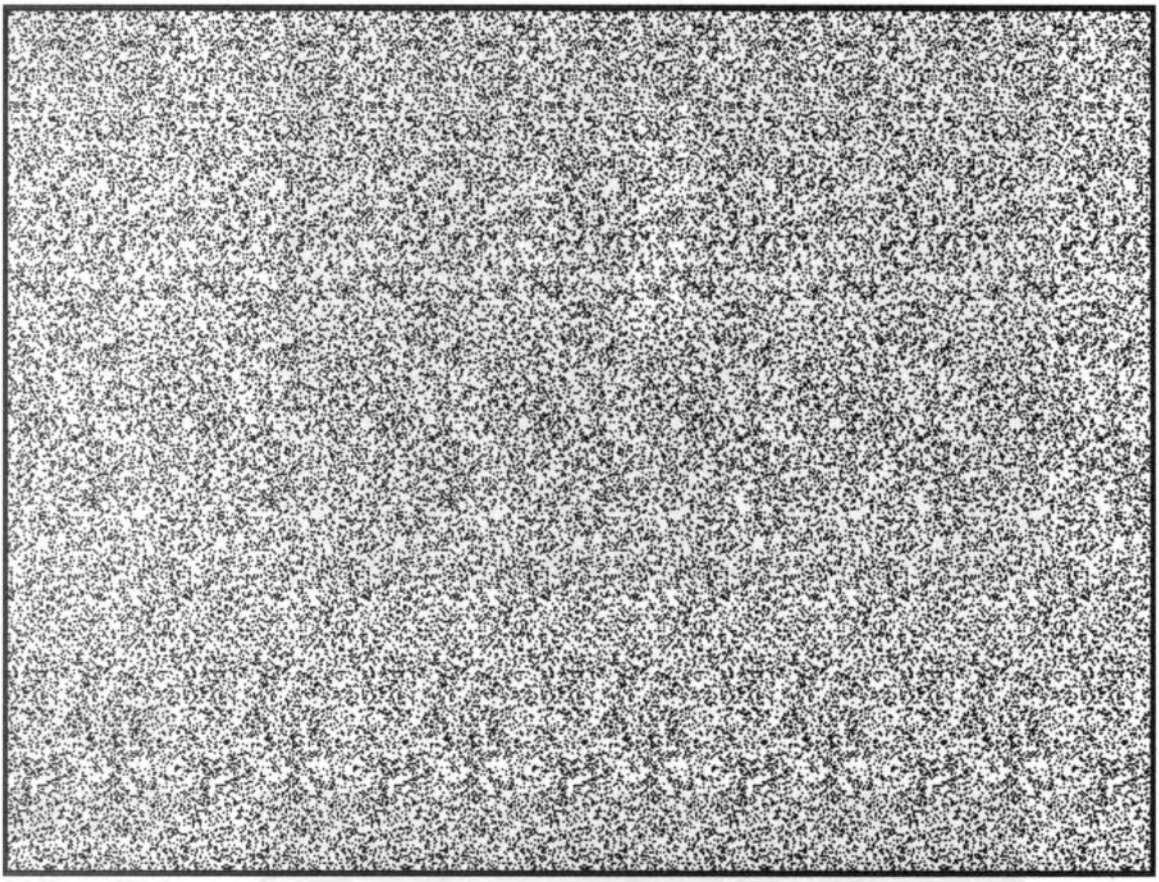


Figure 8.9

Curved surface effect.

Observe that when you use all 16 levels of depth, the object has a nice round look. When you use POPOUT-PRO (the registered version of POPOUT) you can create objects with up to 256 depth levels. With 256 depth levels, the output quality improves considerably.

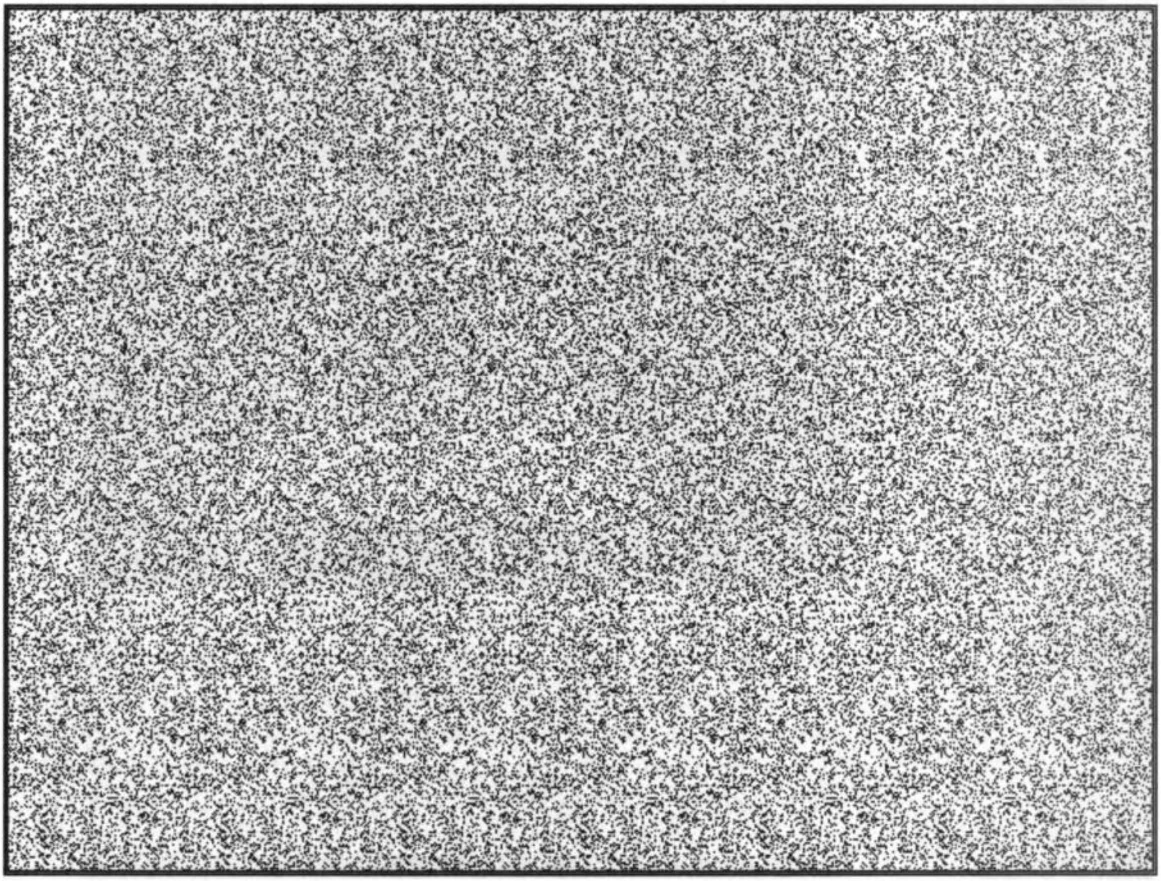


Figure 8.10

Round object RDS.

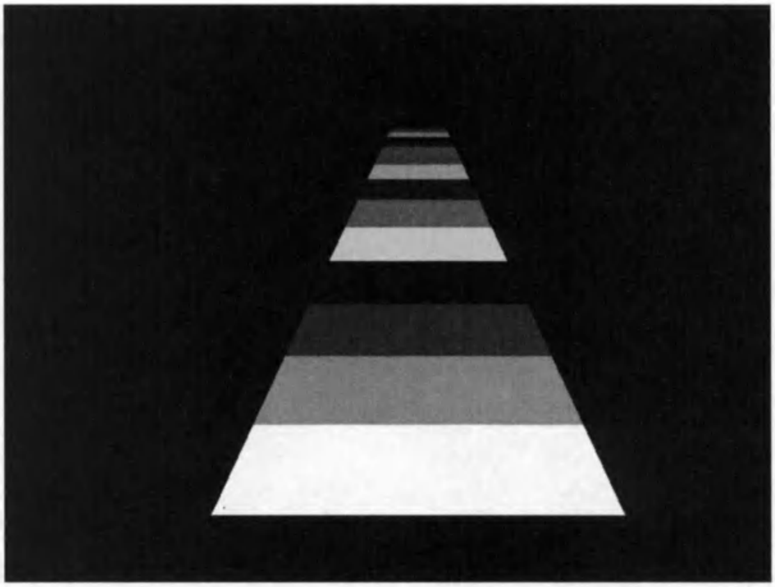
Making Surfaces Appear Infinite

Another simple effect that can be created is that of a surface that goes into the distance forever. The bottom section is nearer to the observer. As you look at the surface from bottom to top, the surface appears to be farther away until it goes into the back plane of the screen.

You can implement this effect in two ways: you can make the surface appear to bend into the back plane or just use a flat surface. To improve the effect, you draw the surface so that the nearest point to the observer has the fullest size. As it goes further and further into the screen, you taper it off in size. In two dimensions, the surface appears to be similar to a triangle or a triangle with the tip cut off (see fig. 8.11).

Figure 8.11

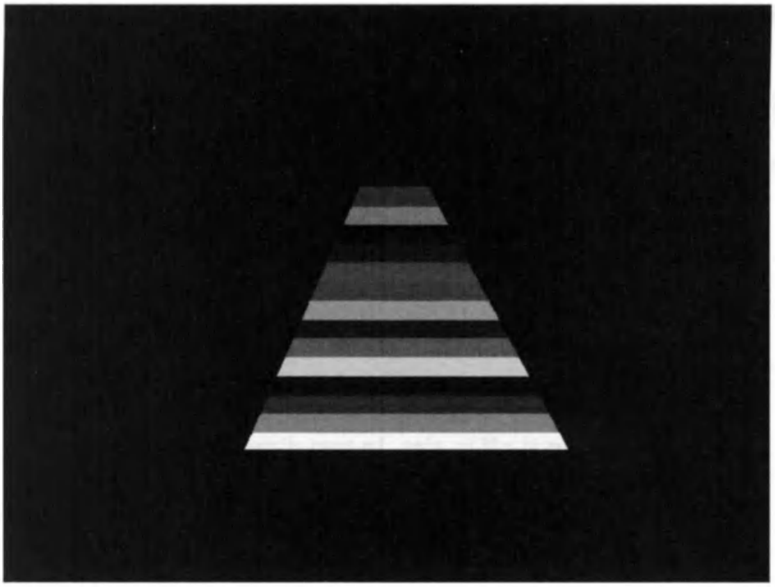
The flat surface appears to curve and go into the page.



When applying colors to the areas, start at color 15 (nearest to you) and choose colors with decreasing numbers (from the palette) as you move up the surface. To create the curved effect, reduce the region's size. When you do, the region appears to go further into the distance (refer again to fig. 8.11). Figure 8.13 displays the RDS of figure 8.11.

Figure 8.12

A flat surface that appears to go into the page.



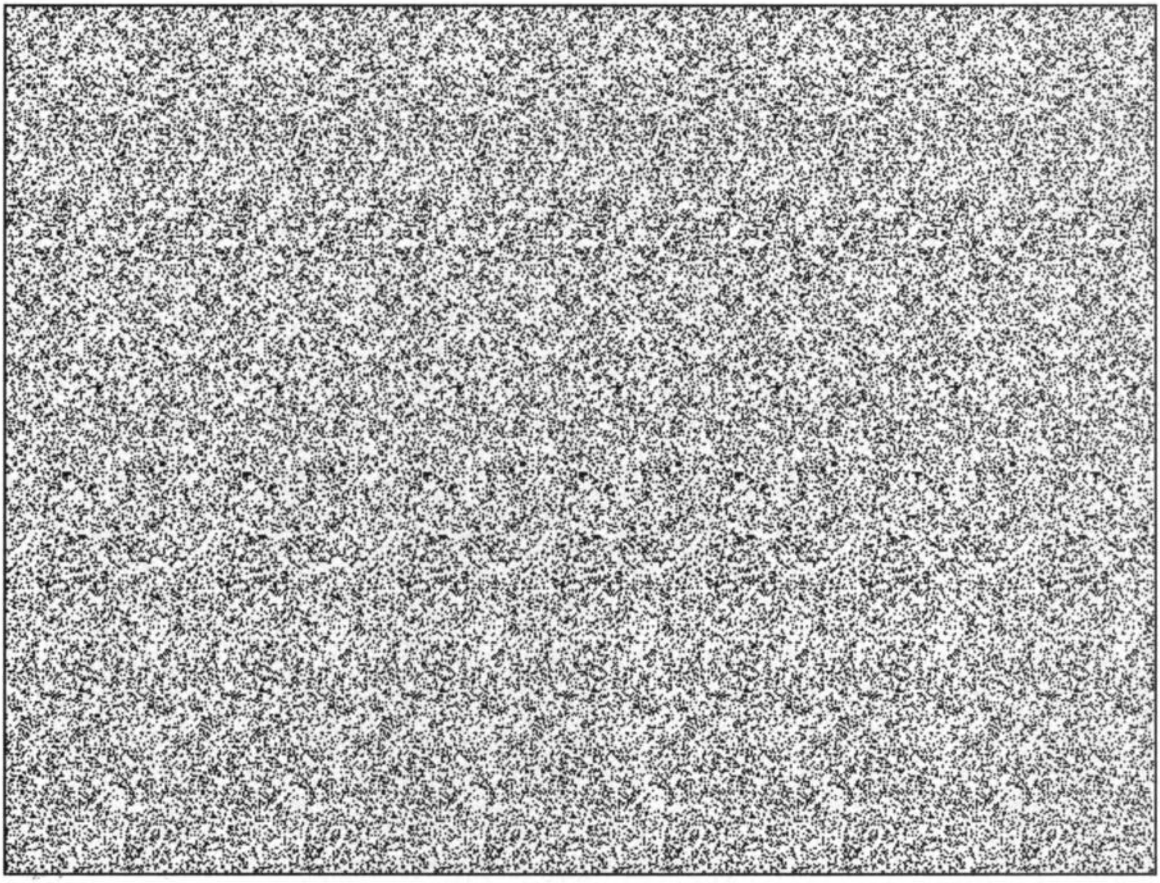


Figure 8.13

Curved surface RDS.

To implement the same effect, use a flat surface that starts near you and eventually ends up going through the back plane of the screen. Redraw the same triangle type of object and subdivide it into regions with the same height (see fig. 8.12). This subdivision maintains a flat surface appearance. Figure 8.14 displays the final RDS.

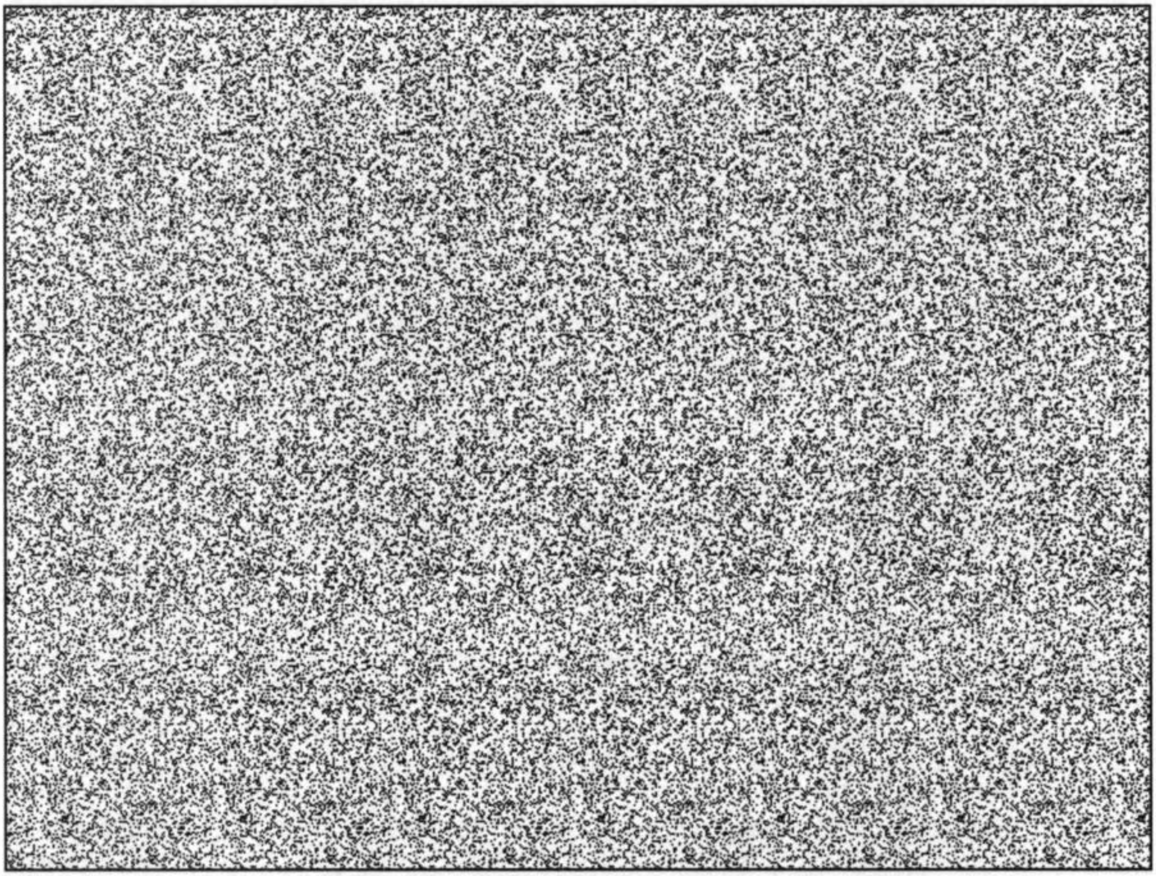


Figure 8.14

Flat surface RDS.

Creating Common Objects

Using the basic tools discussed in the previous section, how can you develop more complex objects? This section explains how to use the techniques you just learned to create rounded objects such as cylinders and flower vases.

Making a Cylinder

Figure 8.15 displays the type of image you need to create for a cylinder.

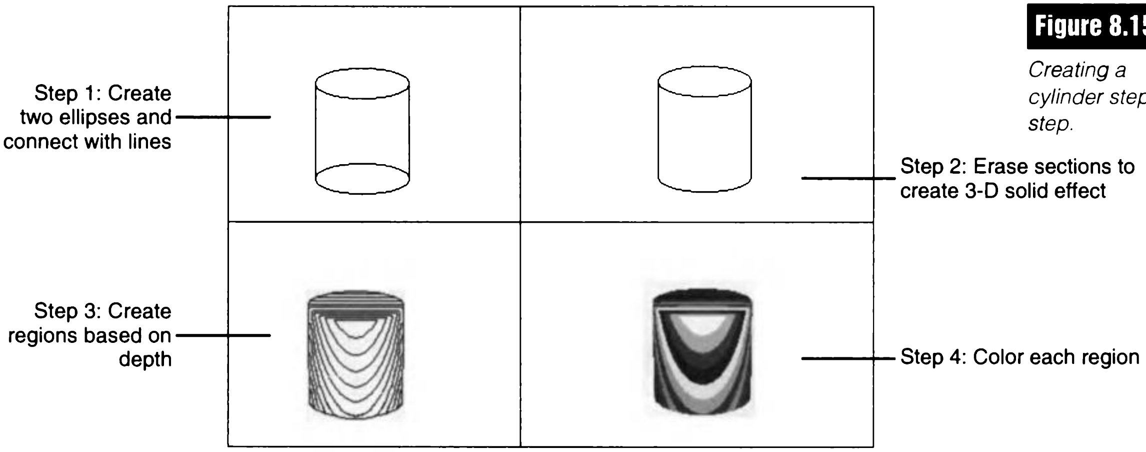

Step 1: Create two ellipses and connect with lines
Step 2: Erase sections to create 3-D solid effect
Step 3: Create regions based on depth
Step 4: Color each region

Figure 8.15

Creating a cylinder step-by-step.

Let's summarize the steps needed to develop the cylinder:

1. Draw a hollow circle that's stretched horizontally to become an ellipse. The ellipse will give the cylinder the effect of being tilted towards you.
2. Copy the ellipse, and drag the copy so that it's placed two inches or so below the original ellipse, ensuring that the left and right edges are vertically aligned.
3. Draw a line connecting the left edges of the two ellipses.
4. Draw a line connecting the right edges of the two ellipses.
5. Imagine that this is now a solid 3-D cylinder. What would you have to erase from it? Part of the bottom ellipse has to be erased.

This completes the outline of the cylinder. The next step is tricky. Create regions in the object by imagining how it would look if you could slice it with a knife. Subdividing this cylinder takes some practice and with some planning you can get good results.

Finally, fill in the regions with the colors, always starting from front to back. This will give you enough depth levels to get a good RDS output (see fig. 8.16). Color plate 9 displays the cylinder partially filled with colors and color plate 10 has the completed image.

Figure 8.16

Full-size cylinder.

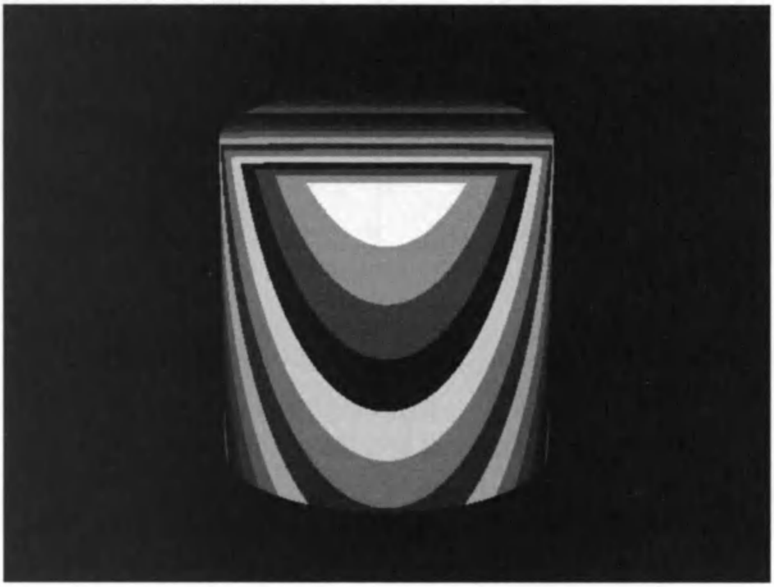


Figure 8.17 presents an RDS of the cylinder just described (see also color plate 11).

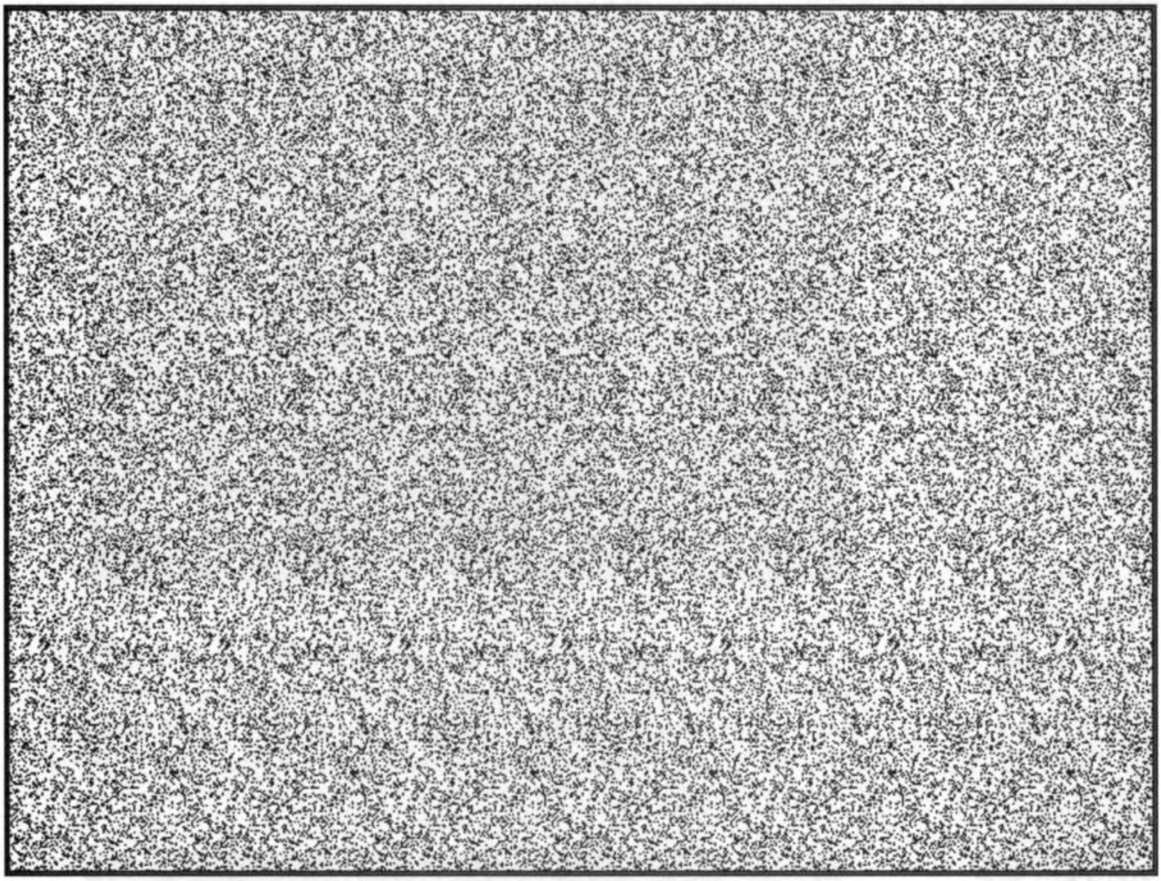


Figure 8.17

Cylinder RDS.

Creating a Flower Vase

Creating a flower vase (without the flowers) is a procedure similar to that of creating a cylinder. The difference resides in that the vase has a varying radius from top to bottom. Again, planning is always necessary and a little experimentation can help you learn how to improve the image. Figure 8.18 displays an outline of the flower vase to be used as the depth file.

Figure 8.18

Flower vase.

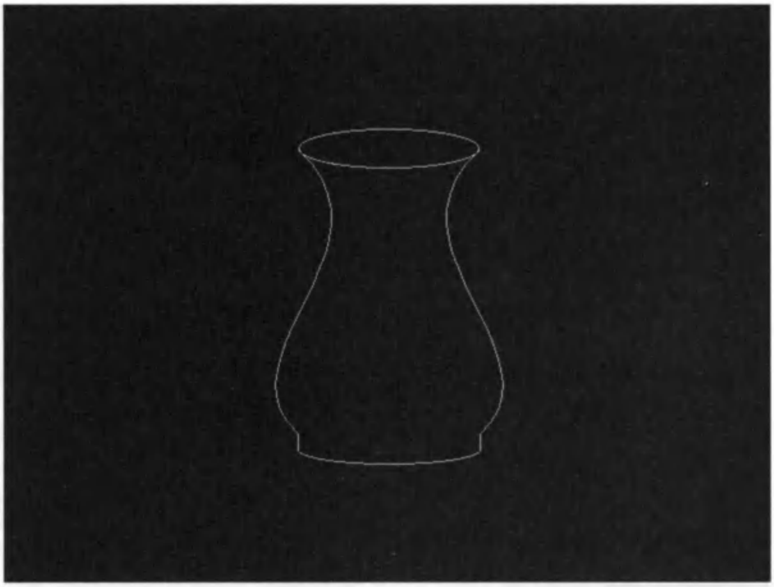
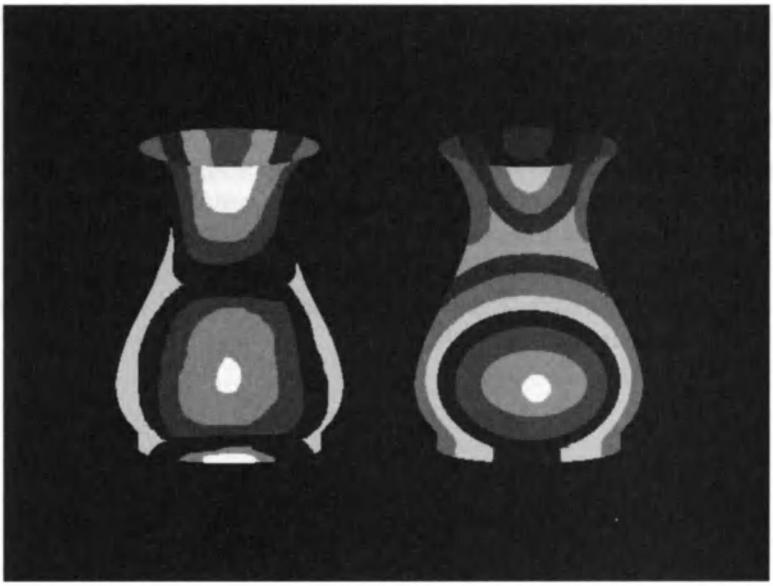


Figure 8.19 displays two different ways of creating and filling the regions. The vase on the left was created by using the brush to draw the regions and the one on the right was created using ellipses and curves. Figure 8.20 shows figure 8.19 converted to an RDS with POPOUT-LITE.

Figure 8.19

Two flower vases with color/depth information.



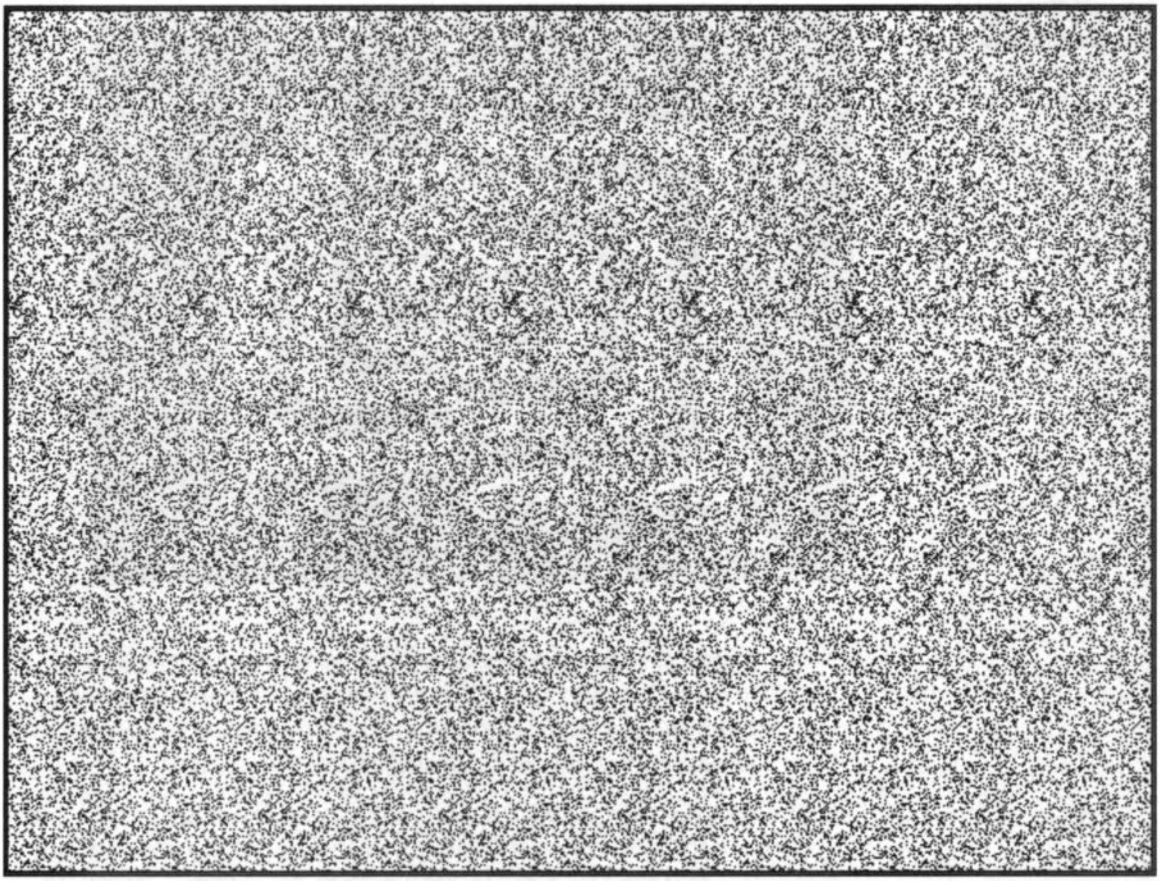


Figure 8.20

Flower vase RDS.

Using Depth Contrast

In this section, you'll experiment a little with contrasting different levels in an object. The idea is to create interlaced regions where depth information is opposite one to the other. Let's take a quick look at a few examples.

Example #1: Two Level Object

Start by creating a set of boxes with rounded corners in a pattern similar to that of figure 8.21. Use as many boxes as you find appropriate.

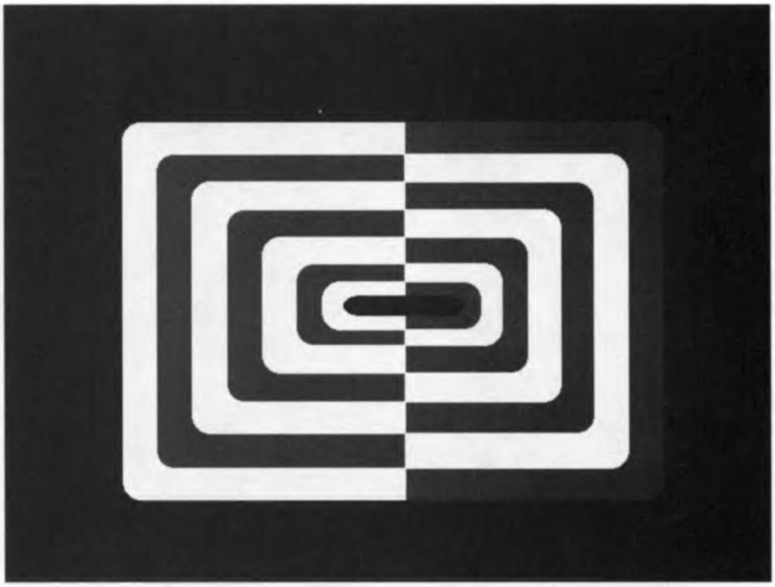


Figure 8.21

Two-level depth object with contrast.

Now, create a line that divides the whole set of boxes through the middle. Using the roller tool, fill each region interchanging between two different levels. I used colors 9 and 15. The further the colors are from each other, the greater the distance between regions.

Figure 8.22 displays the RDS. Observe that both levels are far apart in apparent depth. To decrease their apparent depth difference, choose two colors that are also close to one another, for example, colors 14 and 15.

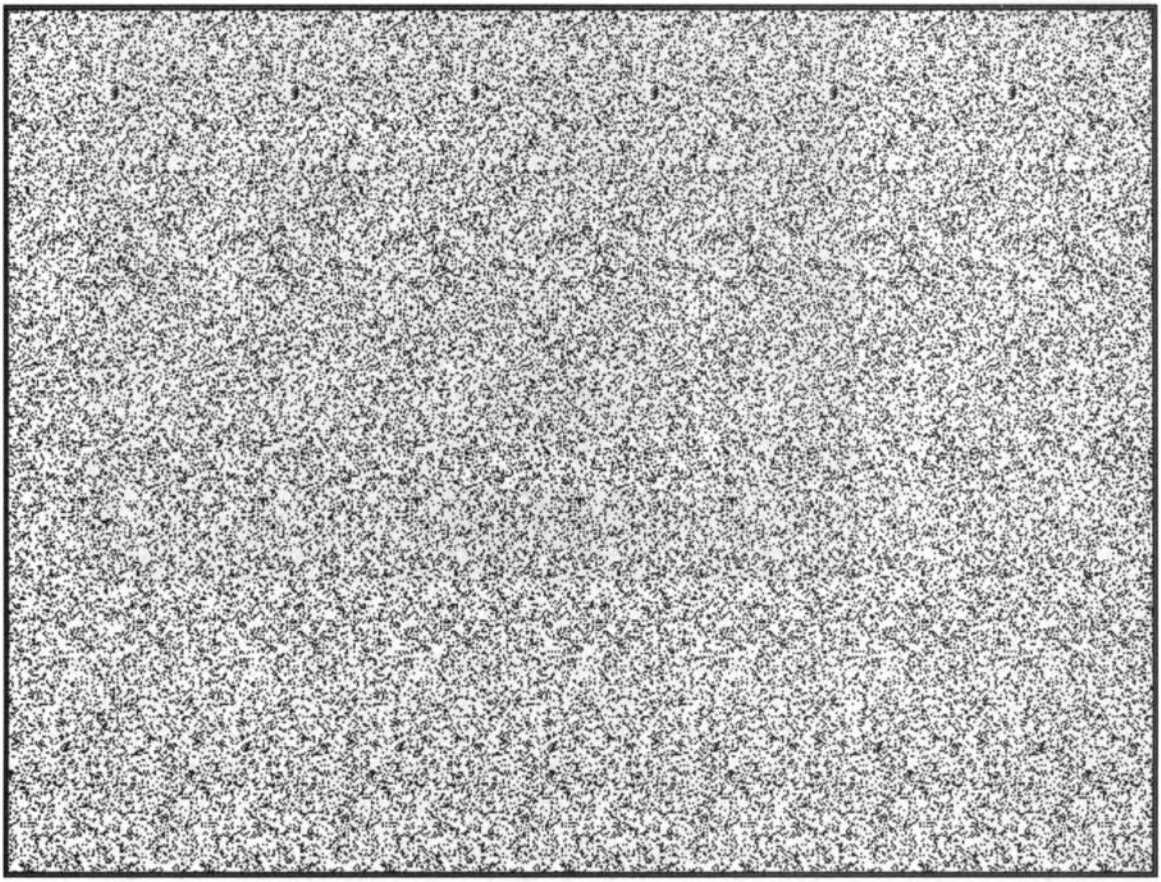


Figure 8.22

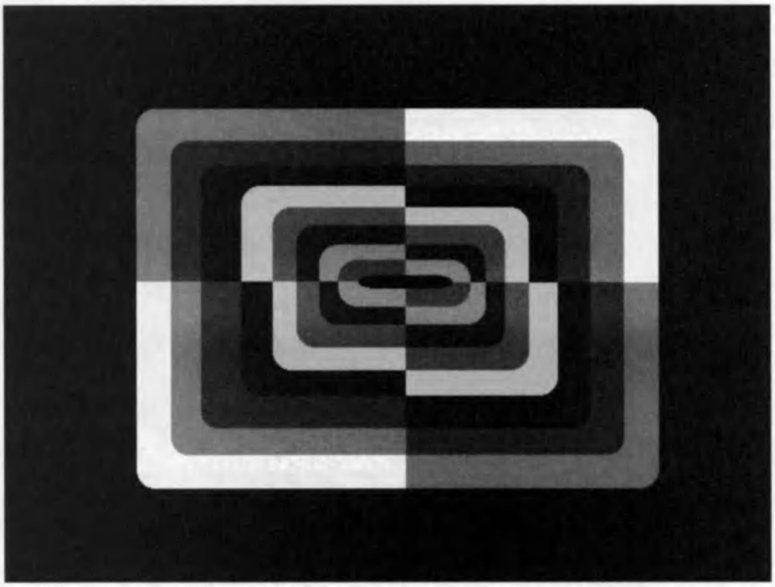
RDS of contrasting object.

Example #2: Multilevel Object

Let's apply the same concept to multiple levels. To create the multilevel object, start by creating a set of boxes similar to what you saw in the previous section. Once this set of boxes is done, create two lines (one horizontal and one vertical) to divide the object in four quadrants or sections (see fig. 8.23).

Figure 8.23

Multiple level drawing with depth contrast.



Use the roller tool to fill the regions. For every band from the center, you'll have four regions to fill. Use any two colors in the band, and contrast them. Now move to the next band and use one of the previous colors and a new one. Continue the process until all the regions are filled. Your drawing should look similar to figure 8.23.

Figure 8.24 displays the final RDS. When you start observing the object, it seems to be similar to a pyramid with the apex going away from you into the distance. With a little time, you'll find out that some of the regions contrast with others in the same apparent level.

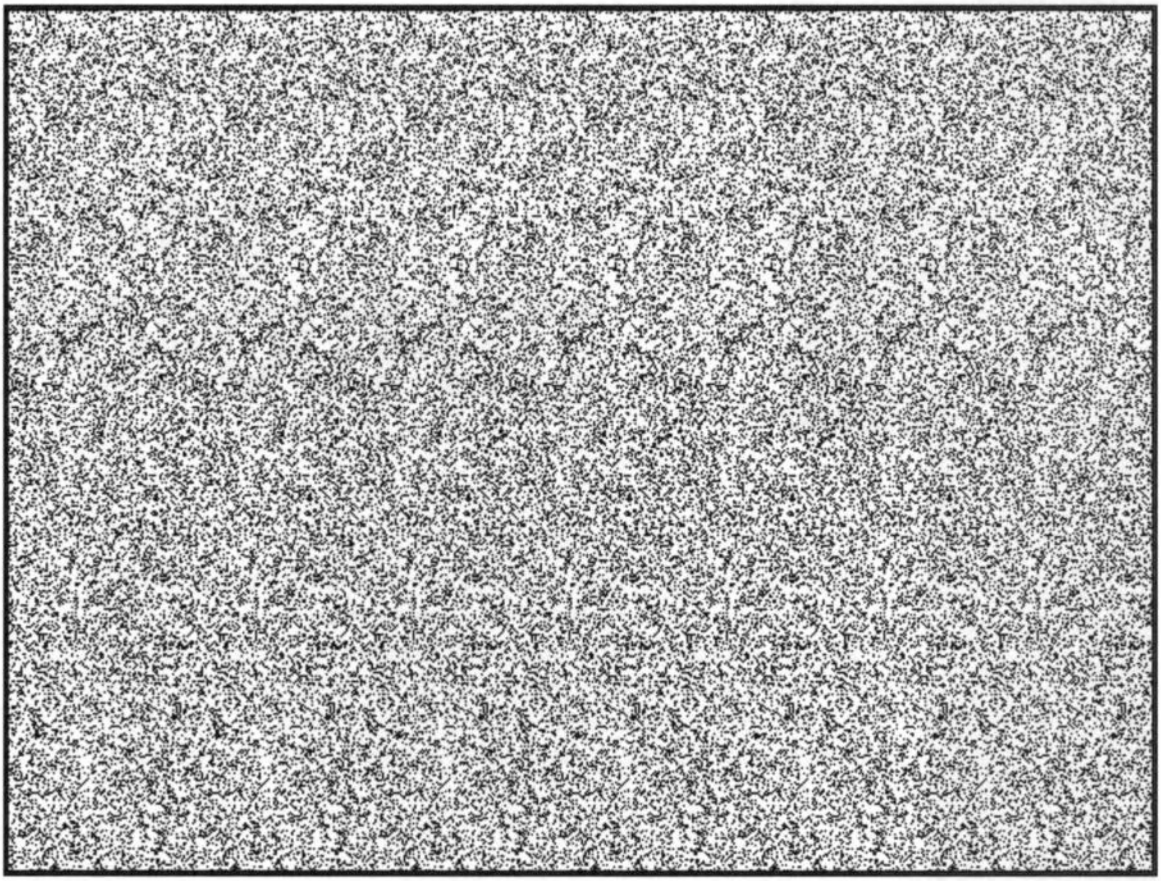


Figure 8.24

RDS of multilevel image with depth contrast.

Summary

Let's stop now. You should have a few ideas that you can experiment with. Let your imagination go when you create the images!

In this chapter, you learned the following techniques:

- How to create flat and curved surfaces.
- How to create the effect of going into the screen's back plane.
- How to create round objects such as cylinders and flower vases.
- How to create contrasts between two or more levels.

9

Using POPOUT-LITE

POPOUT-LITE is the program included with this book that generates random dot stereograms from drawings that you create. POPOUT-LITE was designed to be used with Paintbrush or some similar drawing program and is capable of generating stereograms with up to 16 levels of depth. All the previous examples presented in chapters 7 and 8 were developed with Paintbrush and POPOUT-LITE.

> **NOTE:** POPOUT-LITE is distributed following the shareware concept. If you register POPOUT-LITE, you'll receive POPOUT-PRO, a more advanced version of POPOUT.

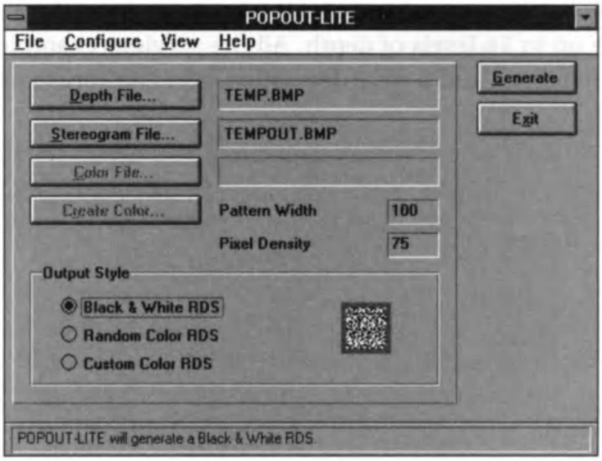
POPOUT-LITE can create three different types of stereograms:

- Black & White RDS
- Random Color RDS
- Custom Color RDS

The program is extremely easy to use; the next sections present its features.

Main Menu Options

POPOUT-LITE's main menu consists of four options (see fig. 9.1). Let's discuss them briefly.



Depth File... TEMP.BMP

Stereogram File... TEMPOUT.BMP

Pattern Width 100

Pixel Density 75

Output Style

● Black & White RDS
○ Random Color RDS
○ Custom Color RDS

POPOUT-LITE will generate a Black & White RDS.

Figure 9.1

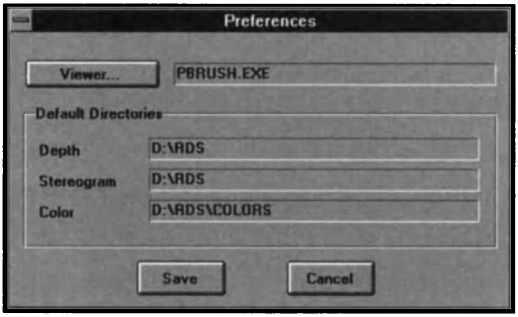
POPOUT-LITE's main window.

File

The **F**ile menu simply enables the user to exit the program. Any file interaction is performed from the main screen.

Configure

Selecting the **C**onfigure menu enables the user to define certain preferences. Selecting the **P**references option from this menu displays a Preferences dialog box (see fig. 9.2).



Preferences

Viewer... PBRUSH.EXE

Default Directories

Depth D:\RDS

Stereogram D:\RDS

Color D:\RDS\COLORS

Figure 9.2

Preferences dialog box.

Use this dialog box to fill in information concerning default directories for your drawings and stereograms. For example, I have my stereograms in the D: drive under the \RDS directory. If you have created custom colors, I suggest you use the Color text box to specify a subdirectory to save them to. For example, I specified the directory \RDS\COLORS to hold my color choices.

POPOUT-LITE defaults to Paintbrush as the image viewer. If you wish to define an alternative program, click the Viewer button and specify the program you want to use.

View

The **V**iew menu enables the user to preview the **D**epth File (that is, the drawing you created with Paintbrush), and the **S**tereogram File (the RDS created by POPOUT-LITE). Choose B**r**owse, and the View Image dialog box appears for you to choose the file you want to see.

Help

As in any standard Windows application, click the **H**elp menu to receive further information on any area of the program.

Selecting Options from the Main Screen

The most used commands in POPOUT-LITE can be accessed from the main screen by pressing the corresponding button.

Depth File

The **D**epth File button lets you select the drawing file to be converted into an RDS (you created this image in Paintbrush). A Depth File dialog box appears on-screen, permitting you to select any 16-color Bitmap (*.BMP) file.

NOTE: Keep in mind that Paintbrush saves PCX drawings with 256 colors as the default. When you create your images always remember to save them as 16-color Bitmap (BMP) images.

Stereogram File

The **S**tereogram File button enables you to specify a new file name for the RDS to be created. You can also use an existing file name after the program asks if you wish to replace it. If you accept, POPOUT-LITE will overwrite the older file with the new RDS file.

Color File

Color files are color schemes that have been custom designed and can be used in the process of generating random dot stereograms. Use the **C**olor File button when you want POPOUT-LITE to create a stereogram with a color scheme that you have previously defined and stored on your disk as a color file (see the next section, "Create Color," to learn how to do this). The **C**olor File button is enabled only when Output Style is set to Custom Color RDS.

Color Files have a default extension of .POC. POPOUT-LITE includes a sample color file named USFLAG.POC that describes the color scheme of Red, White, and Blue.

Create Color

By selecting the C**r**eate Color button, you can access the Create Color File dialog box (see fig. 9.3). Use this dialog box to create a new color scheme and save it as a color file.

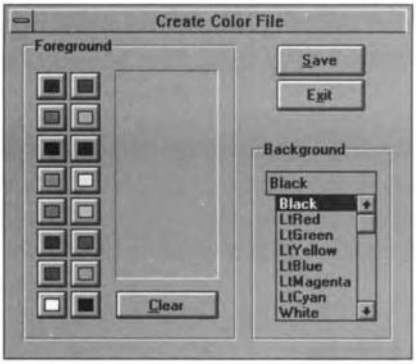


Figure 9.3

Create Color File dialog box.

You can use any color file with any image in order to create an RDS. The color file basically is a list of foreground colors and a background color that is used to generate the final RDS. For example, the file USFLAG.POC has only three colors: LtRed, White, and LtBlue. When an image is generated with this color scheme, the colors appear spread out from top to bottom on-screen. In other words, the top of the RDS starts with the LtRed color, slowly converts into White, and finally changes to LtBlue.

If you've got the Output Style set to Random Color, this dialog box is also used to help POPOUT-LITE with its selection of colors. If you select a Background Color, POPOUT-LITE generates an RDS with random colors, but tries to bias the selection toward the color you've chosen. If you press any of the colored Foreground Button Colors, POPOUT-LITE creates the RDS by using only these colors. By pressing these buttons more than once, you can specify the exact mix of colors. For example, if you press the Red button three times, press White twice, and then press Blue once, the resulting stereogram consists of only Red, White, and Blue dots. Furthermore, there are twice as many White dots as Blue dots, and three times as many Red dots as Blue dots.

Specifying a Pattern Width

POPOUT-LITE uses a random pattern that is repeated across the image. The distance between the repeating patterns is referred to as the *pattern width*. When POPOUT-LITE generates a stereogram, it reads this parameter to determine how many pixels to generate before repeating the pattern. The default value is 60 pixels, but a better value is 100.

NOTE: You may find it helpful to review the first few chapters briefly, particularly chapter 5, for information about pattern width and color density.

Pixel Density

The Pixel Density setting is a percentage value that specifies the ratio of foreground to background pixels that should be generated. The default value is 50 percent, which tells POPOUT-LITE that when it creates a Black & White or Custom Color stereogram, it should make roughly half of the pixels the foreground color, and half the background color. The Pixel Density setting can't be changed when Output Style is set to Random Color RDS.

NOTE: When Output Style is set to Random Color RDS, POPOUT-LITE randomly chooses all pixel colors. This random selection has the negative effect of not letting you define the foreground and background colors.

The Pixel Density value comes in handy when you need to print stereograms. By adjusting the density value, you can control how light or dark the RDS appears on the printer. For example, high density levels reduce the number of background pixels considerably, almost to the point of creating a "minimalist" RDS (that is, it has the least amount of information necessary for the brain to recognize the hidden object).

Output Style

POPOUT-LITE can generate three types of RDS images: Black & White, Random Color, and Custom Color RDS. Color images are saved as 256-color .BMP files.

Black & White images use less disk storage space than color images. If you intend to print your RDS with a black-and-white printer, there's no need to create the colored RDS.

Random Color RDS images are created by choosing a random set of colors as opposed to Custom Color RDS images, which use the Color File option to impose a particular color scheme on the output image. These colors can be chosen either by the user, or by POPOUT-LITE.

Generating a Random Dot Stereogram

Let's present the complete cycle of using POPOUT-LITE, step-by-step:

1. Use the Windows Paintbrush accessory to create a drawing similar to that displayed in figure 9.4. Draw each letter in a different color.

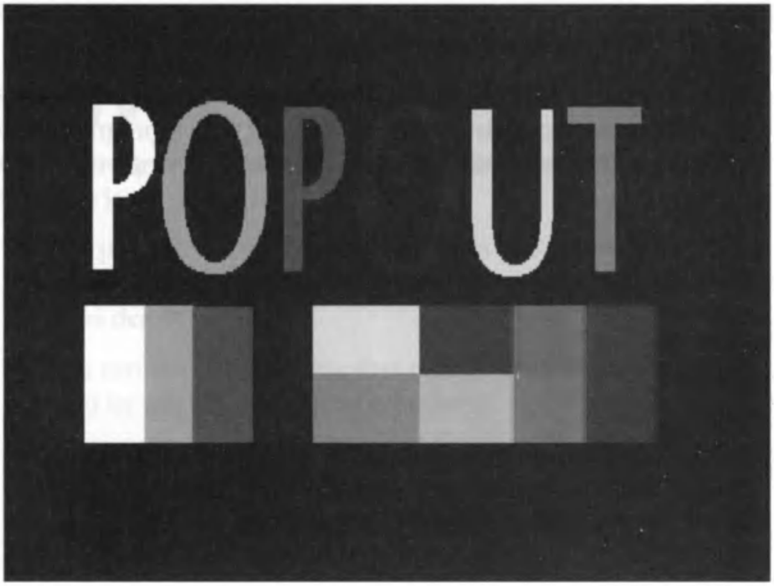


Figure 9.4

Creating a sample drawing.

2. Use the **F**ile **S**ave command. Type **TEST** in the File **N**ame box. Click the down arrow to drop down the Save File as **T**ype list and click the 16 Color bitmap options. Click OK to close the dialog box and save the file.
3. To run POPOUT-LITE, return to Program Manager, double-click the POPOUT-LITE program group icon, and double-click the POPOUT-LITE program icon. The program starts and the main screen appears.
4. Click the **D**epth File button. To select the file TEST.BMP, select the drive and directory where it's stored and then click OK.
5. Click the Stereogram File button and type **TESTRDS** in the File **N**ame box. Choose a new destination drive and directory, if needed, using the appropriate lists. Click OK when you've finished.
6. Select an Output Style by clicking the appropriate style (choose Black & White RDS in this case).
7. Set the Pattern Width to 100 and the Pixel Density to 75. To change one of these settings, double-click the current value to select it, and then type a new number.
8. Click the **G**enerate button. After a few seconds (depending on your computer's speed) the RDS is created. Let's take a look! (see fig. 9.5).
9. Select the **V**iew menu. Click **S**tereogram File.

At this point, POPOUT-LITE uses the information on the default viewer program (Paintbrush) to find out which program to run. To see the full RDS, select the **V**iew menu from the Paintbrush and click **V**iew Picture.

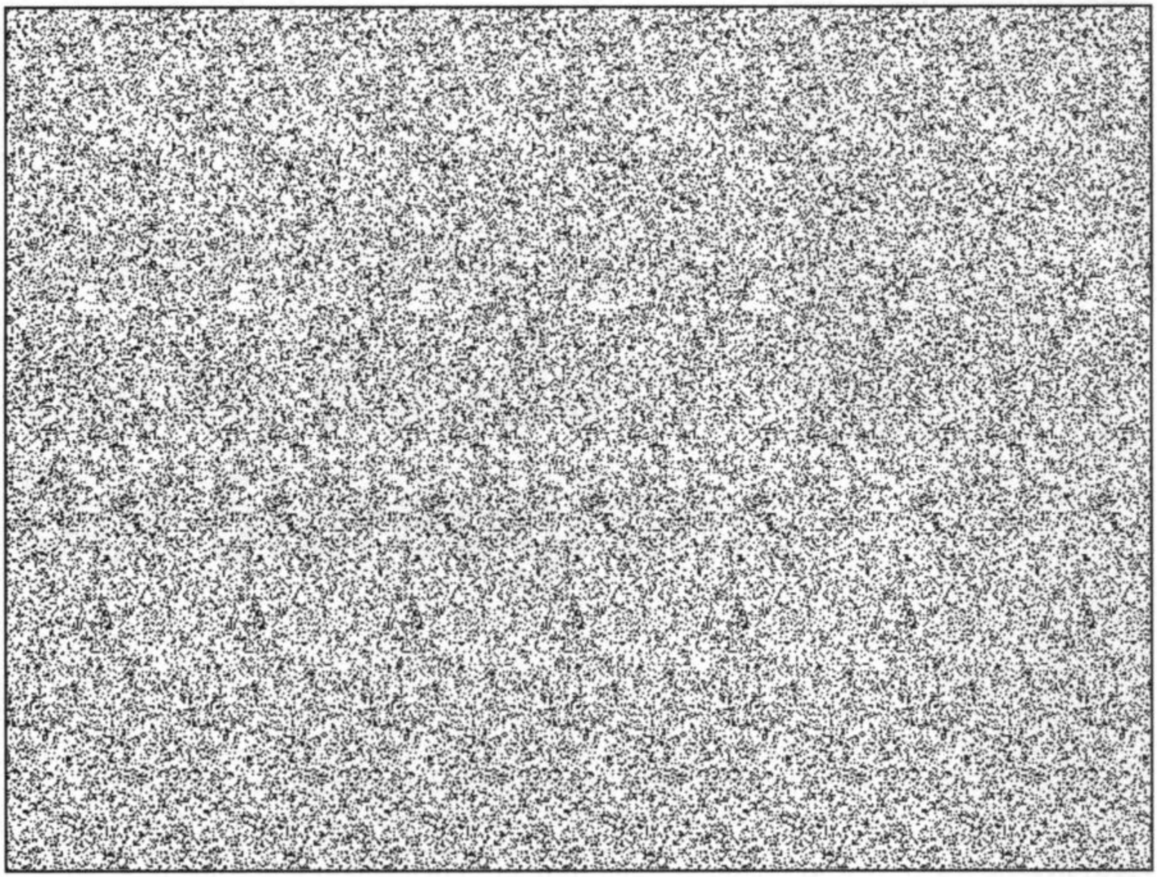


Figure 9.5

RDS Image of TEST.BMP

Summary

This chapter presented the different features available in POPOUT-LITE. You learned that:

- The program can generate three types of RDS images and that the program gives the user control over the pattern width and pixel density.
- You can select a color file that consists of a list of colors to be used by the program for RDS creation.
- The process of loading an image and converting it into an RDS file is straightforward and fast. You also learned the steps for converting any 16-color BMP image to a stereogram.

10

Converting and Using Random Dot Stereograms

This chapter covers a series of topics that will help you get the most from RDS development. It takes a look at everything from where to find depth image sources to how to give others access to your depth images and RDSs. In addition, the chapter presents various utilities that you can use to convert depth images between different formats. Lastly, you'll discover what to keep in mind when developing depth images with programs other than Windows Paintbrush.

Using Existing Depth Images for Your Stereograms

Most of the time, you will develop your own RDSs from scratch. Using Paintbrush, you will draw the depth image and feed it to POPOUT-LITE. Some depth images take a long time to develop. This doesn't always have to be true, as you learn in the next section.

Sources of Depth Images

A good number of RDS fans are out there who share their ideas and depth images. These images can be found in a wide variety of file formats that then need to be converted to a format readable by POPOUT-LITE. Let's discuss some important sources for images.

Accessing Computer Bulletin Board Systems (BBSs)

If you have a modem and a communications program, you can access most bulletin boards (BBSs) in your area. If you don't know any local BBS phone numbers, talk to friends and co-workers and find out if they know of a listing of phone numbers for BBSs in your area. Chances are that you will find a list of BBSs in your area or the whole USA (from which you can identify those BBSs nearest you).

TIP: *Boardwatch* magazine, available at many bookstores and computer stores, regularly publishes listings of public access BBSs.

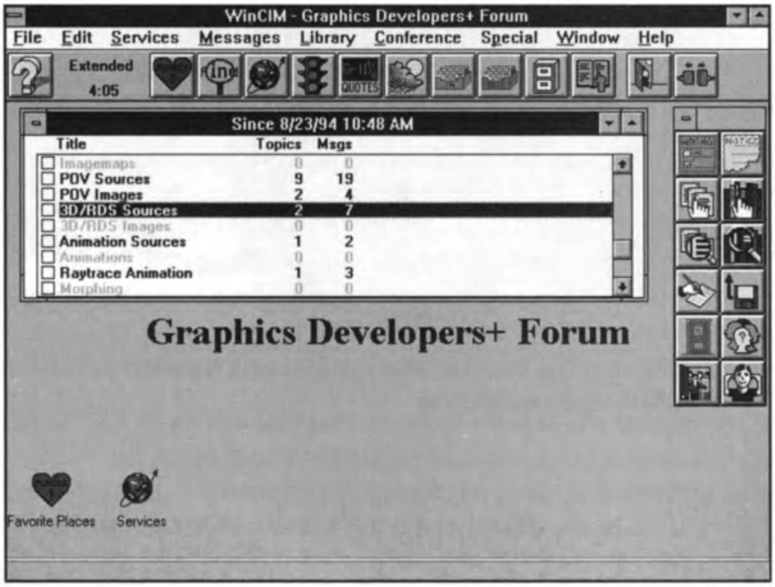
Most of these BBSs have message areas where different topics are discussed among BBS callers. Check for areas that are based on RDS topics or computer graphics. Once there, you can ask for more specific information concerning RDS sources and related information. Normally, each message area in a BBS has a corresponding file area under the file section. These areas are used by the BBS operators and the callers to post and download files. There you can find text files, depth images, RDS programs, and related utilities that you can either download freely or for a small fee.

If your local BBS does not have a computer graphics or RDS section, try suggesting the idea in an e-mail message to the BBS operator. In addition, you can ask other BBS users about the idea of implementing an RDS section. Ask them if they know of any other places where RDS utilities and depth files are available.

Finally, check around for a FAQ (Frequently Asked Questions) document. Most people get tired of seeing and answering FAQ. The solution to this problem is to post a text file that addresses all the FAQ and is updated on a weekly or monthly basis. This resource is one of the most powerful ones out there.

Exploring On-line Services

On-line services such as CompuServe, America Online, Delphi and others have discussion forums for almost every topic. For example, inside the GraphDev (Graphics Developer's) forum, CompuServe has an area totally dedicated to RDSs and resources. Once you access this using GO GRAPHDEV, you can interact with other RDS fans through e-mail or by chatting through the modem on a one-to-one basis or in a group discussion. Figure 10.1 displays the GraphDev forum RDS message area and figure 10.2 shows the file sections.



Since 8/23/94 10:48 AM

Title	Topics	Msgs
□ Imagemaps	0	0
□ POV Sources	9	19
□ POV Images	2	4
□ 3D/RDS Sources	2	7
□ 3D/RDS Images	0	0
□ Animation Sources	1	2
□ Animations	0	0
□ Raytrace Animation	1	3
□ Morphing	0	0

Figure 10.1

RDS message area in CompuServe.

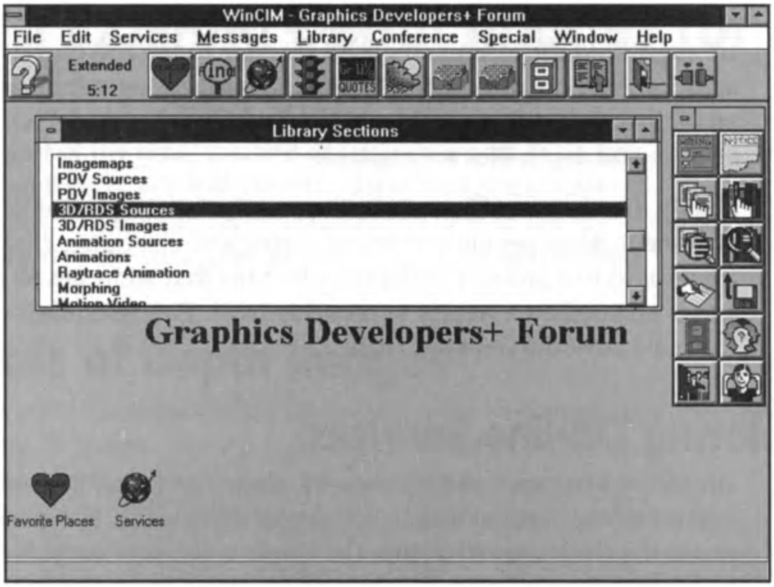


Figure 10.2

RDS file section in CompuServe.

Most of these on-line services have start-up kits that you can find at your local software store. These kits usually will give the customer a trial period where you can access the system as a visitor. Take advantage of the opportunity to browse around and find areas that interest you.

Jumping onto the Internet

If you have access to the Internet, a "network of networks" connecting more than 20 million computer users, look for addresses related to RDS topics. A good starting point is in the comp.graphics USENET group. Posting a message there can get you started. Usually, you'll find 100 to 3J0 messages a day on any given topic; be prepared to do some keyword searching. If you find an interesting message, reply to the author of the message and ask for more detailed information on other RDS sites and sources.

Scanning Images

What if you don't have a modem? Another way to generate images to be later used for RDS is through the use of image scanners. A scanner is a device that converts a document or photo into electronic form. Once this image is saved, you can convert it to whatever format is the most appropriate.

NOTE: Many service bureaus and printers can scan images for a fee. Make sure the printer saves the file to a PC disk (not Mac) in the proper formats (PCX or BMP will work, as Paintbrush can read either).

The scan image doesn't give any depth information; you must add it yourself. How do you do this? Refer to chapter 8, where you learned different techniques to make objects appear three dimensional with Paintbrush. Let's take, for example, figure 10.3.

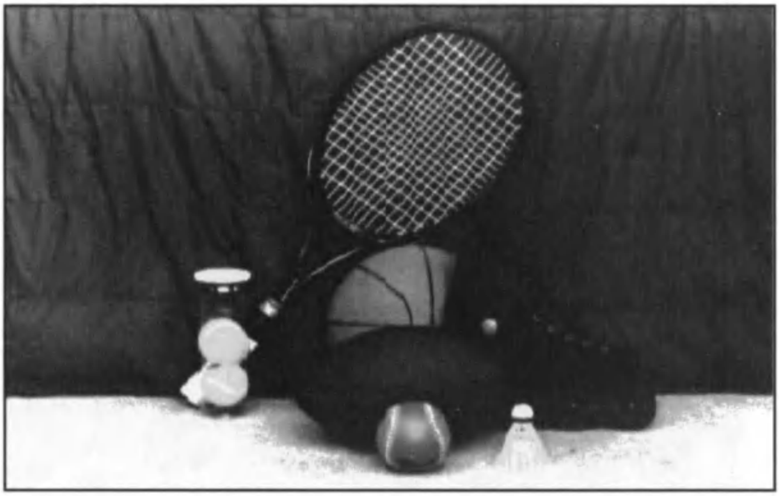


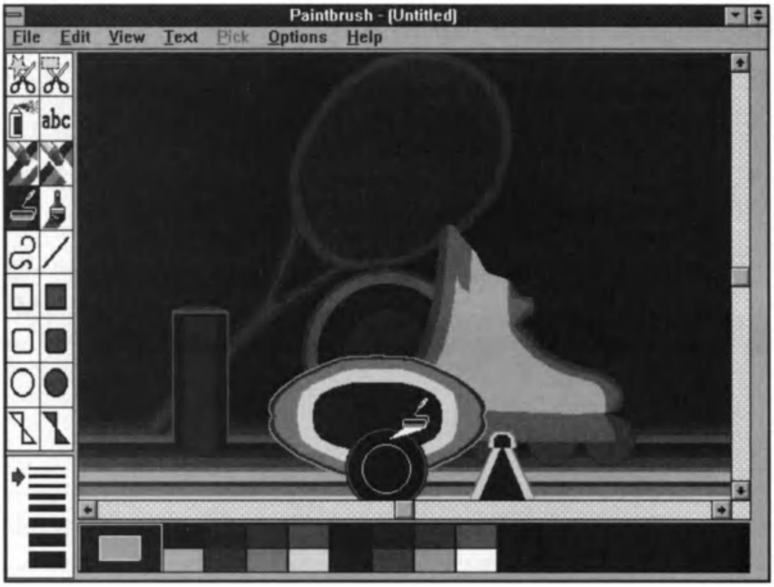
Figure 10.3

Scanned image of sports equipment.

Figure 10.3 displays a scanned image of some sports equipment. This image is converted into a bitmap and loaded into Paintbrush. Using the techniques explained in chapter 8, you can start drawing on top of the image itself to create a 3-D effect (see fig. 10.4).

Figure 10.4

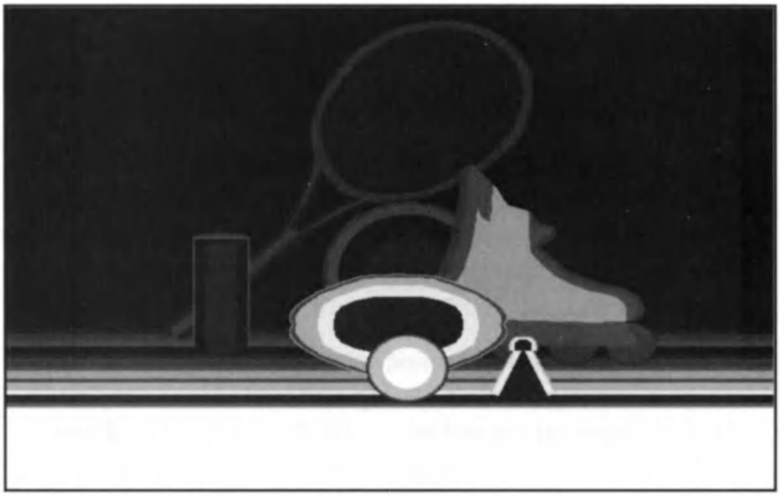
Scanned image with Paintbrush colors on top.



You may wish to delete some of the objects that you do not want to appear in the final RDS. To delete them, define cutout regions with the Scissors or Pick tools and choose **E**dit Cu**t** (Ctrl+X). Figure 10.5 displays the completed depth image. Observe how all the depth levels were created and the shapes simplified.

Figure 10.5

Final depth image of scanned photo.



You can save the image as a 16 color BMP file and use POPOUT-LITE to convert it to an RDS, as shown in figure 10.6.

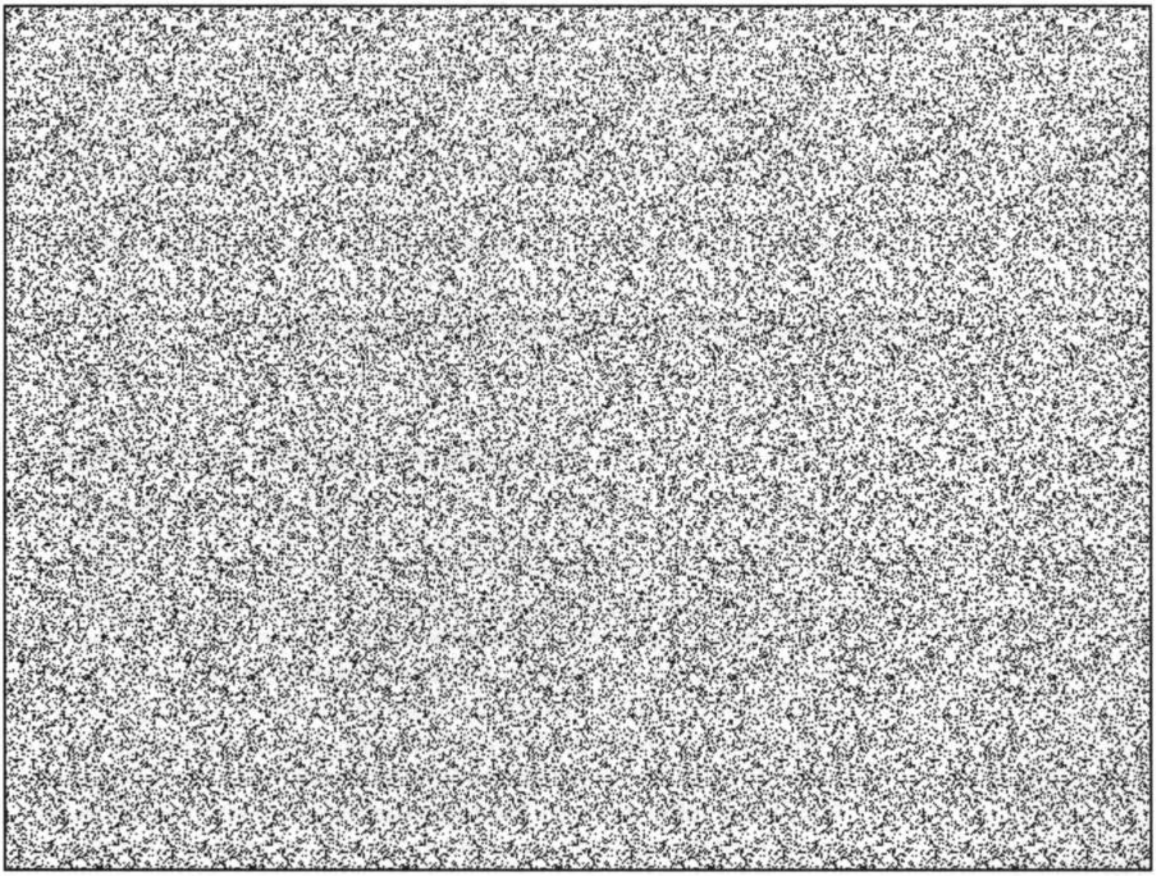


Figure 10.6

RDS of sports equipment.

Understanding Depth Image Formats

When you talk about depth image formats, you must differentiate between the format of how the file was saved and how the depth information is saved in a given file. Let's discuss both.

Working with File Formats

File formats include formats such as TGA, BMP, PCX, GIF, TIFF, JPEG, and others. Formats by themselves do not give any information concerning the depth of the objects in the depth file. The RDS program must be able to understand and decode the format as necessary. Once the information is decoded, the RDS program must make sense of the depth information stored in the colors or shades indicated by the data.

As you work more and more with different file formats, you'll find out that some formats are easier to work with than others. This is not because the format may be difficult to read or interpret, but because the number of utilities available may be limited. In addition, some file formats such as TIFF tend to take up huge amounts of disk space. We'll discuss conversion utilities in more detail later in the chapter.

Understanding Depth Information

As I mentioned previously, file formats are independent of depth information. Once the file has been decoded, the RDS program is ready to analyze the depth information contained in the data itself.

There are two commonly used depth formats. The first one is based on information saved as grayscale colors. In other words, the further an object is from the observer, the darker the color will be and vice versa. For example, if the palette has 16 levels, each point in the depth image must be defined with one of the grayscale levels between 0 and 15. These grayscale colors are independent of whatever color the object originally had and are only a function of the depth. Take a photo of a yellow tennis ball, for example. After scanning the photo, you have a BMP image of it that you can load into a paint program. Once you apply grayscale levels (by drawing over the original image), no part of the tennis ball will remain yellow, only some level of gray.

The other depth format available is based on replacing the colors in a given pixel with the depth information. For example, a program that saves depth encoded images may save each pixel as a 24-bit value: 8 bits for the Red (R) component, 8 for the Green (G) component and 8 for the Blue (B). This gives us a total of 24 bits for saving depth information.

The depth encoding algorithm may take the R and G spaces and set the B equal to zero. In these R and G spaces it will save the depth information as a 16-bit number (8 bits of R + 8 bits of G = 16 bits). In other words, the depth information is independent of the color that eventually shows up. The RGB spaces of each pixel, in reality, save depth information, not a color.

These methods are used by the developers of RDS programs and are not manipulated directly by the persons that wish to generate stereograms. These formats are not really standards but are some of the most commonly used formats. Other RDS developers may decide to use totally different formats or the same with small variations. For example, in the grayscale format someone may decide to use the darkest color to represent being near to the observer while the brightest means that the object is extremely far. On using depth encoding, the developer may decide not to use the R and G components of the color but use G and B instead.

Be sure that the utilities being used to convert between formats are compatible, or you can lose the depth information forever. The conversion utilities that come with the book are specifically designed for POPOUT.

Converting Depth Images for POPOUT-LITE

POPOUT-LITE can only read 16-color BMP files. This requires that depth images saved in any other format be converted first. For example, if you have an image downloaded from an on-line service which saves all files as GIF images, you should convert it to a TGA file. To do this, you need to run a conversion program. Once the depth image is a TGA you may use the utility TGA2BMP, which is included with this book. This utility was developed specifically for POPOUT-LITE and POPOUT-PRO (the registered version of POPOUT) and should be used instead of any other conversion programs.

The syntax for TGA2BMP is

```
TGA2BMP in_filename.TGA  out_filename.BMP color_num
```

where `color_num` is either

`16` (POPOUT-LITE)

`256` (POPOUT-PRO)

(To run TGA2BMP from the DOS prompt, C:\>, type the command line and press Enter, or use **F**ile **R**un in Windows.) Imagine you have the image SOURCE.TGA and want to convert so it can be read with POPOUT-LITE. The name of the output image will be OUTPUT.BMP.

To perform this conversion, type in the following command line and press Enter:

```
TGA2BMP SOURCE.TGA OUTPUT.BMP 16
```

> **NOTE:** When converting TGA to BMP files, use the utility TGA2BMP, which was developed specifically to work with POPOUT's LITE and PRO versions.

Personalizing the Depth Image

Once the image is in BMP format, you may personalize the image by adding, deleting, or modifying the existing objects. Recall chapter 7, which explains how to define cutout regions that can then be copied and pasted, deleted, or simply moved to a new location.

Think also of adding your name or the name of the person you'll give the RDS to. This is an excellent media for private messages.

Making Your Depth Images Available to Others

Now that you've created your depth file you may want to share it with other RDS fans. You can either give your depth files to friends on disk, or upload the image to a BBS, on-line service, or the Internet. An important point to have in mind when sharing depth images is this: you don't know what RDS or paint program the other person will be using. Therefore, it's important to save the depth file in a well-known image format.

To convert 16-color BMP images to TGA grayscale, you should use the utility BMP2TGA, which is also included with this book. To convert, for example, SOURCE.BMP to GRAY.TGA, type the following command line at the DOS prompt and press Enter (or use **F**ile **R**un in Windows):

```
BMP2TGA SOURCE.BMP GRAY.TGA
```

BMP2TGA was also developed specifically for BMP images that are compatible with POPOUT-LITE. Once you convert the BMP to a TGA file, you should convert it to a standard file format (if it's different from TGA). This may be necessary due to the fact that some BBS operators and on-line services accept only certain types of file formats. For example, CompuServe accepted only GIF for a long time. In other words, the system will impose a certain image format that you must adhere to. Again, use a conversion program to go from TGA to any other format.

NOTE: An excellent conversion program is PaintShop Pro from JASC, Inc. which is distributed as shareware. Contact JASC, Inc. at 10901 Red Circle Drive, Suite 340, Minnetonka, MN 55343. Voice (612) 930-9171 or FAX (612) 930-9172.

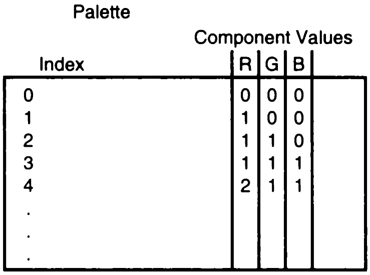
If it's possible, post additional information concerning the file that you have posted either as e-mail in the system or as a description line inside the depth file, which some formats accept. Give plenty of information to make using your depth file easy to use and provide an address so that users can reach you for more information.

Using Paint Programs Other Than Paintbrush

If you don't have access to Windows Paintbrush, you may decide to use a different paint program to create depth images. There are a few points to remember if you eventually decide to use them with POPOUT-LITE.

Color Palette Concepts and Tricks

A color palette can be visualized as an array of colors. Each color is defined by a triplet, <R,G,B>, and the location of each triplet is identified by an index (see fig. 10.7). If the palette has 16 colors, the index varies from 0 to 15. Each of the components of the given color has information related to the resulting color. If the palette has 256 colors, the index varies from 0 to 255. Most of the paint programs let you edit the palette in a similar fashion to the Paintbrush Edit Color dialog box and even load different palettes.



Palette

Index	Component Values R	 G	 B	
0	0	0	0	
1	1	0	0	
2	1	1	0	
3	1	1	1	
4	2	1	1	
.				
.				
.				

Figure 10.7

Simple representation of color palette.

A problem arises when you have grayscale levels in an image. Imagine trying to add depth to an image with 256 grayscale levels. The colors are so near to each other that it is difficult to figure out what is in front of what. In other words, it becomes increasingly difficult to differentiate between levels. To overcome this problem, you may wish to use a palette that has broken down the various levels into large groups. Each of these groups will consist of different variations of a given color. For example, if you have 16 grayscale levels, you can divide the levels as shown in the following mini-table.

Index	Color
0	Blue
4	Red
8	Yellow
12	White

From looking at this table, you know that the color Blue is further than the Red, Red is further than Yellow, and White is the nearest color to the observer. There are only four levels defined at this point, so you can break down colors in between the defined levels using variations of the given colors, as shown in the following table:

Index	Color
0	Blue
1	Blue #1
2	Blue #2
3	Blue #3
4	Red
5	Red #5
6	Red #6
7	Red #7
8	Yellow
9	Yellow #9
10	Yellow #10
11	Yellow #11
12	White

Now all levels have been defined in the palette. The colors Blue #1 through #3 indicate different variations of the original color Blue. Still, we know that *any* of the blue colors will still be farther than *any* of the red colors and so on. This palette can be easily saved and used for other drawings.

To work with the palette, load it before starting any drawing. Once the drawing is done, change the palette to the grayscale and save the depth file. The program will replace the corresponding colored values with grayscaled ones. This simplifies the development of depth images.

Color versus Index Depth Encoding

In the previous section, I presented briefly what a palette is and how information is saved. A related topic is how depth information is read from a palette.

Some RDS programs, such as POPOUT-LITE, use color-based depth encoding. This means that POPOUT-LITE attaches a specific depth to a specific color. For example, color <0,0,0> may indicate that this point is the furthest possible to the observer. Color <15,15,15> may indicate to POPOUT-LITE that this point is the nearest to the observer. A point in between could be defined as <8,8,8>, and so on.

Observe that the index information was not used at any time to help in the calculation of depth information. On the other hand, index depth encoding uses the index number of a given color to define levels. In other words, index #3 can have any possible color and this will never affect the depth. The color is simply not used. The RDS program would go pixel by pixel and investigate what is the index of the given color in the pixel. Once it's found, the depth information is used to generate the level.

You must be careful that when images are converted from one format to another, no information gets lost. If, for example, the relationship between colors and indices is not maintained, index-based depth encoding will not work. If the conversion program modifies the palette colors even slightly, the color-based depth encoding will not be able to find the colors, thus the RDS output will be corrupted.

The choice of index or color-based depth encoding is done by the RDS program developer. For POPOUT-LITE, you should focus on color-based depth encoding.

File Format Requirements for POPOUT-LITE

POPOUT-LITE used the color-based depth encoding approach. Recall that before developing any drawings in Paintbrush, the palette POPOUT.PAL must be loaded and the final file must be saved as a 16-color BMP file before conversion to an RDS. The POPOUT.PAL colors have already been selected as defining certain depth levels. If you use a different palette, POPOUT-LITE may provide unpredictable results when you create the RDS image.

Using A Ray Tracer for Depth Image Creation

As I mentioned earlier in this chapter, you don't necessarily have to create every depth image with Paintbrush. In order to demonstrate a different technique, we'll cover briefly how POLYRAY can be used to create the depth image.

What is POLYRAY?

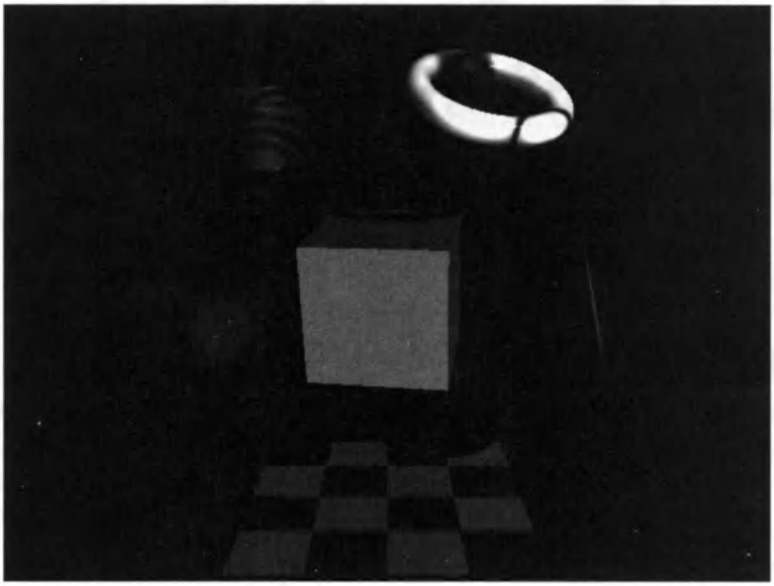
POLYRAY is a shareware ray tracing program developed by Alexander Enzmann. A ray tracer creates photo-realistic images. That is, it uses mathematical methods to model the behavior between a light source and an object. If you have access to CompuServe you can download POLYRAY from the GraphDev forum.

Observe that items around you do not have the same surface characteristics. For example, a drinking glass can be transparent or it can be colored; sand paper appears to be rough at a distance and covered with small specks; wood displays a certain color and you can see veins running across it. A ray tracer takes all this information into consideration and generates a photo-realistic image based on the surface characteristics you define for an object. This information is usually defined in a text file that the ray tracer reads and uses to generate the final image. Figure 10.8 shows a ray-traced image created with POLYRAY.

NOTE: POLYRAY is an extremely powerful program that requires a book by itself. If you wish to learn POLYRAY in more detail, check the book *Adventures in Ray Tracing* by Alfonso Hermida, published by Que. This book covers POLYRAY from top to bottom.

Figure 10.8

Ray-traced image created with POLYRAY.



Listing 10.1 presents the data file used to create figure 10.8.

Listing 10.1 POLYRAY Data File for Figure 10.8

```
include "colors.inc"
include "texture.inc"

viewpoint {
 from      <0, 1.0, -5>
 at        <0.0, 0.0, 0.0>
 up        <0.0, 1.0, 0.0>
 angle           45
 resolution      640,480
 aspect          1.33
 }

// LIGHT_SOURCE
light <0,  0,  -100>
light <50, 50, -50>

//PLATE
object {
    polygon 4, <-1,0,-1>, <1,0,-1>, <1,0,1>, <-1,0,1>
    translate <0,-1.5, -0.2>
    texture {
      checker matte_white, matte_black
      scale <0.5, 0.5, 0.5>
    }
}

//BOX
object {
    box <-1, -1, -1>, <1, 1, 1>
    scale <0.5, 0.5, 0.5>
    rotate <-10, 10, 0>
    shiny_green
}

//CYLINDER
object {
    cylinder <0,0,0>, <0,0.4,0>, 0.4
    rotate <-15,0,0>
    translate <-0.8,1.0,0>
    wooden
}

//TORUS
object {
    torus 0.5, 0.1, <0,0,0>, <0,1,0>
    rotate <-30,0,-10>
    translate <0.7, 1.4, 0>
    white_marble
}

//CONE
object {
    cone <0.0, -1, 0.0>, 0.5, <0.0, 0.0, 0.0>, 0.0
    translate <1.4, 0.48, 0>
    shiny_blue
}

//SPHERE
object {
    sphere < -0.96, 0, -1.12>, 0.25
    shiny_red
}
```

Let's look at a few of the components to give you an idea of what's going on in listing 10.1. The first important component in creating a POLYRAY data file is the creation of a *viewpoint.* A viewpoint definition tells POLYRAY where the observer of a given scene is located. For example, the keywords `from`, `at`, and `up` tell POLYRAY where the observer is standing relative to a 3-D coordinate system <X,Y,Z>, what is being looked `at`, and what is the `up` direction relative to the observer. The following section of the data file defines the viewpoint information.

```
viewpoint {
 from       <0, 1.0, -5>
 at         <0.0, 0.0, 0.0>
 up         <0.0, 1.0, 0.0>
 angle            45
 resolution       640,480
 aspect           1.33
 }
```

The `angle` keyword indicates that the observer has a "field of view" angle of 45 degrees. The `resolution` of the final images is 640 × 480 and the aspect ratio of the image is 1.33 (equal to 640 divided by 480).

Now, let's look at the simplest object definition in the file. The sphere is created using the following section of code:

```
//SPHERE
object {
    sphere < -0.96, 0, -1.12>, 0.25
    shiny_red
}
```

The line `//SPHERE` is just a comment and it's used to tell the person writing the data that a sphere definition follows next. The keyword `sphere` indicates that the object is a sphere and that the center is located at `<-0.96, 0, -1.12>` in the x, y, and z coordinates, respectively. The radius of the sphere is 0.25 units. Finally, the surface texture is shiny and the color is red. The keyword `shiny_red` is just a name and tells POLYRAY to look at the definition for more detailed information of how to create a red and shiny surface.

As I mentioned earlier, POLYRAY has lots of different commands that go beyond the scope of this book. You don't need to know everything about how to use it, but you should know how to use an image created with POLYRAY for RDS development if you desire to use those kinds of images.

Using POLYRAY Images with POPOUT-LITE

To create images that can later be used with POPOUT-LITE, you must tell POLYRAY to generate a depth file. This can be done by including certain command line parameters when generating the image from POLYRAY. For example, to create figure 10.8, I used the following command line parameters (run POLYRAY from a DOS window): `POLYRAY DEMO.PI -V 1 -W -t 0`. In order to create a depth encoded file you must add the parameters `-u` and `-p z`. The final command line should look like this:

```
POLYRAY DEMO.PI -V 1 -W -t 0 -u -p z
```

DEMO.PI is the name of the file presented in listing 10.1.

POLYRAY creates a file named OUT.TGA that is depth-based. To convert it into a POPOUT-LITE compatible depth file, you must go through two steps.

The first step is to use the included DOS utility PLY2POP.EXE. This program reads the POLYRAY depth file and converts it to a grayscale image, where black pixels represent points that are furthest from the observer, and white pixels represent the points nearest to the observer. The command line to use from the DOS prompt is:

```
PLY2POP OUT.TGA DEPTH.TGA
```

where `OUT.TGA` is the depth file created by POLYRAY and `DEPTH.TGA` is the output file in Targa format.

The second step for converting the image to a POPOUT-LITE compatible depth image is to use TGA2BMP to convert DEPTH.TGA to a bitmap (BMP) file that is made up of the 16 basic Paintbrush colors. Type the following command line from the DOS prompt and then press Enter:

```
TGA2BMP DEPTH.TGA DEPTH.BMP 16
```

Now you're ready! Run POPOUT-LITE and select DEPTH.BMP as the **D**epth Image file. Enter the name TEST.BMP as the **S**tereogram File name. Set the Output Style to Black & White RDS and click **G**enerate. Figure 10.9 is the resulting image. Can you see all the objects?

The process is simple and enables you to generate complex images without the need to draw each object in Paintbrush.

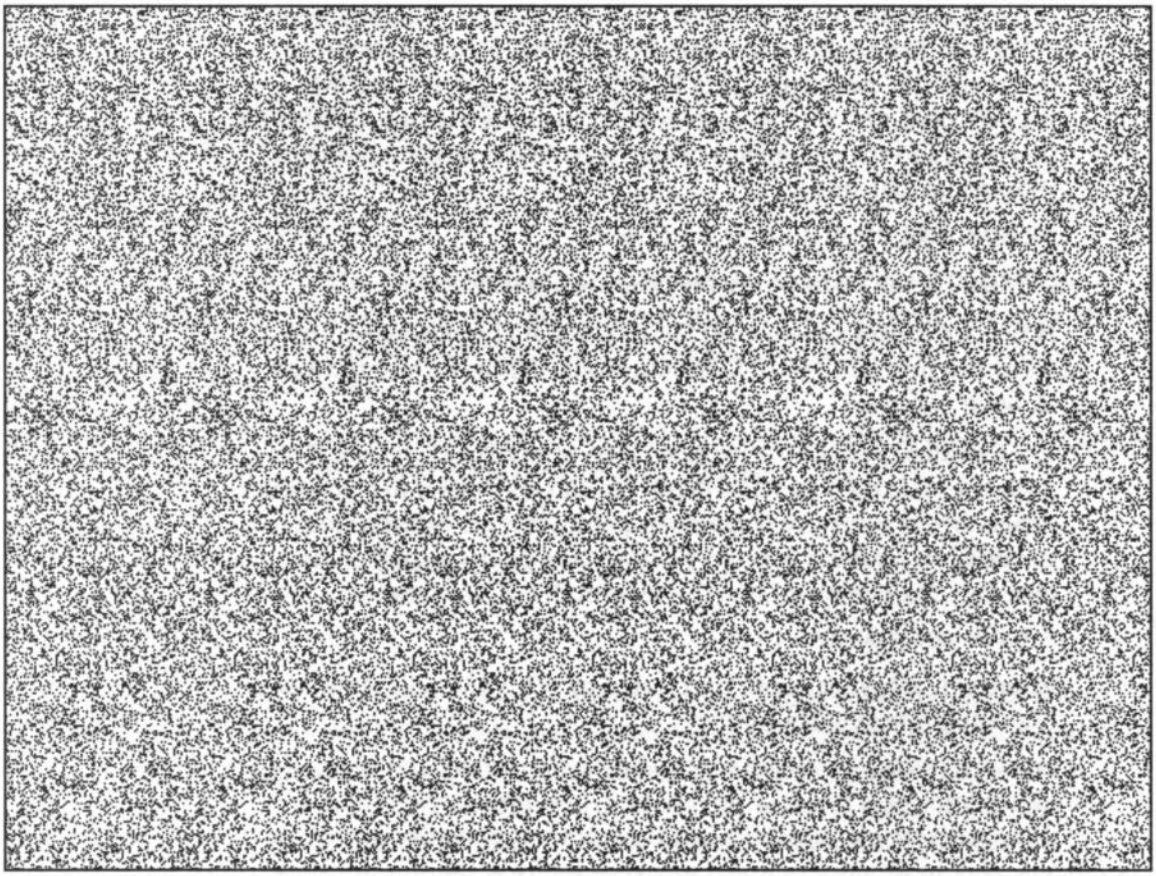


Figure 10.9

RDS of ray traced image created with POLYRAY.

Other Uses for RDS

Once you have developed a set of good RDSs, you may start thinking of taking the whole process to a higher level. Here are some ideas of things you can create with RDSs:

- *Postcards.* You've seen these in card stores!
- *T-shirts.* Do you like people to stare at you?
- *Posters.* One of the best ways to display an RDS.
- *Books.* A few books are already out there that are collections of RDSs. If you have a good set of them, think of the possibility of publishing them either as a booklet or as some sort of magazine.

Lots of people enjoy the challenge of figuring out what the RDS image is about.

- *Animation.* To develop animation, you may want to check out a program named DTA. This program can convert a sequence of images into an animation. This would require that you create a set of RDSs where the objects inside are at different points depending on which image is being generated. You can use these files and DTA to create what's called a *FLI* or *FLIC animation.* Download DTA from CompuServe's GraphDev forum.
- *Windows Wallpapers.* Use the output from POPOUT-LITE to define it as a Windows wallpaper. The process is simple and the BMP format is already compatible with Windows. Start by saving the BMP file in the Windows directory then in the Main program group, double-click the Control Panel icon, and double-click the Desktop icon. In the dialog box that appears, click Wallpaper, select the RDS BMP file, and then click OK.

Printing Your Image

You can print your RDS from Paintbrush. Use the **F**ile **O**pen command and select the file you wish to print. Please remember not to scale the output as this will distort the RDS. If you wish to generate a large print then you must start by generating a larger depth file.

In order to create hard copies of the RDSs (postcards, posters and so on) you can also try contacting service bureaus (printers). These businesses are dedicated to high quality printing and some of them can read disk images. Others can even convert image files to 35mm film in order to create photos or enlargements.

NOTE: The magazine *PC Graphics & Video* includes a "Service Bureau Locator" that includes service bureaus from around the USA.

Summary

This chapter covered lots of different topics that can help you make your life easier once you start developing your own RDSs. Let's review the topics discussed:

- The different sources of depth images and related utilities.
- The difference between file formats and depth formats.
- How to share your depth images with others.
- What to keep in mind if you don't have Paintbrush.
- How ray tracing is used to create depth images.
- What to do with all those RDSs.

Index

Chaos Holograms

Chaos Holograms is a Random Dot Stereogram generation program. Chaos Holograms will create RDS images from any 256 color PCX picture file. You can create computer generated Random Fields or design your own full color Random Fields. Multiple algorithms are supported and are user selectable. Chaos Holograms is available for DOS and Windows users. Chaos Holograms was used to generate the Chaos FX Inc. RDS images shown in this book. By including this advertisement with your order, you will receive a five dollar discount on the purchase of Chaos Holograms. If you order by phone, mention that you saw this advertisement and you will receive the five dollar discount.

Chaos Merchandise

Chaos FX Inc. has a wide range of RDS products available including T-shirts, sweatshirts, tote bags, and postcards. If you are interested in RDS images, please call or write for a free catalog of Chaos FX Inc. merchandise.

Chaos Holograms Software	regular price	$24.95
Chaos Hologram T-shirts	regular price	$19.95
Chaos Hologram Sweatshirts	regular price	$29.95
Chaos Hologram Tote bags	regular price	$12.95
Chaos Hologram Postcards	regular price	$1.00

Please include $3.00 for shipping.

Tell us that you saw this ad and receive a discount on all Chaos FX Inc. merchandise.

Chaos FX Inc.
830 Cedarbrae Ave.
Milton, Ontario
Canada L9T 3X1

Phone (905)875-2519
Fax (905)878-2000

Call or write for your FREE catalog.

You have the capability to create professional quality stereogram images right on your PC!

You can create stereograms that have up to 256 distinct levels of depth, which far exceeds the number of levels in virtually all professionally created images. Create not only Random Dot Stereograms, but also create colorfield stereograms using designs or photograph scans of your own choice! Experiment with a number of adjustments to create images that are just right for you. Let your imagination take you away!

GP POPOUT is the commercial version of POPOUT-PRO, with all the latest enhancements and with features that even POPOUT-PRO doesn't have. Try you own hand with stereogram animation!

GP POPOUT features:

- The same easy to use interface as POPOUT-LITE
- Up to 256 levels of apparent depth
- Black & White, Random, or Custom Color RDS images
- Colorfield and wallpaper stereograms
- Three selectable image generation algorithms
- Grayscale Targa Depth Image support
- Print quality, 24-bit color output capability
- Built-in resizing for poster-sized outputs
- Depth linearity correction
- Depth factor adjustment
- Divergent or convergent stereograms
- Batch capability to support animation
- Smoothing depth file filter that improves colorfield quality
- Optional Guide Dot generation

GP POPOUT retails for $35. Include the serial number of the POPOUT-LITE software and the coupon below to receive a 50 percent discount on GP POPOUT! Are you a potential stereogram artist? Order now!

Installing POPOUT-LITE from the Book Disk

POPOUT-LITE was designed to run on an IBM compatible personal computer. This is a Windows program; you must have Microsoft Windows version 3.1 or better on your computer. The disk included with this book is a 3.5 inch, double-density (DD) floppy. Most PC disk drives are capable of reading a double-density floppy.

To install POPOUT-LITE on your hard disk drive, first carefully remove the disk from the sleeve on the inside of the back cover, and then follow these simple steps:

1. Start Windows if Windows isn't already running on your computer.
2. Double-click the Main program group in Program Manager.
3. Double-click the File Manager program item icon to start the Windows File Manager.
4. In the list of directories at the left of the window for your hard drive (probably C:), scroll up and click the root directory to select it (C:\).
5. Choose **F**ile Cr**e**ate Directory. In the dialog box that appears, type **TEMP** as the directory **N**ame (or enter another name if you already have a TEMP directory), and then click OK.
6. Insert the book disk in drive A: or B: of your computer. Double-click the disk icon for the drive where you inserted the disk at the top of the window for your hard drive. This should open a new window for disk A: or B:, showing that the book disk contains one file, POP.EXE.
7. Click POP.EXE to select it.
8. Click the **F**ile menu, and then click **C**opy.
9. In the **T**o box, type **C:\TEMP**. (TEMP is the directory you created in step 5. If you named the directory something else, substitute that name for TEMP in the command.) Click OK, and File Manager copies the file to your hard drive.
10. In the window for drive C:, scroll down the list of directories with the scroll bar, and then click the TEMP directory (or the directory you just copied POP.EXE to) to select it.

11. Double-click the POP.EXE file. This unarchives the files stored within POP.EXE.
12. Choose **W**indow **R**efresh to see the unarchived files.
13. If you can't see a file named INSTALL.EXE listed at the right side of the C:\TEMP\*.* window, use the scroll bars at the right to display it. Double-click INSTALL.EXE to install the POPLITE program.
14. Follow the instructions of the POPOUT-LITE Install utility. The utility defaults to installing the software on your hard drive at C:\POPLITE, but you may install it wherever you want.
15. When the install utility finishes, it creates a POPOUT-LITE Program Group containing a number of icons. The icons that look like a painter's palette will start Microsoft Paintbrush, and show you some sample stereograms and depth images.
16. To start POPOUT-LITE, double-click the icon that looks like a miniature stereogram. Actually, the icon really is a stereogram. Can you see the letter P?

NOTE: After installing POPOUT-LITE, you can delete the directory you created in step 5 (TEMP, or whatever you named it), and all the files it contains).

By opening this package, you are agreeing to be bound by the following agreement.

This software product is copyrighted, and all rights are reserved by the publisher and author. You are licensed to use this software on a single computer. You may copy and/or modify the software as needed to facilitate your use of it on a single computer. Making copies of the software for any purpose is a violation of the United States copyright laws.

This software is sold as is without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Neither the publisher nor its dealers or distributors assumes any liability for any alleged or actual damages arising from the use of this program. (Some states do not allow for the exclusion of implied warranties, so the exclusion may not apply to you.)

Hidden Images: Making Random Dot Stereograms

3-D stereograms are everywhere, and now you can create your own! With this book/disk set, you'll learn everything you ever wanted to know about these fascinating images.

- Learn what random dot stereograms are and how they work
- Create your own incredible stereograms with the software included
- Discover the best way to view these popular images
- Get the inside story on the algorithms, parameters, and other factors involved in a hidden 3-D image
- Create detailed scenes, random fields, and special effects
- Output your stereogram on screen or printer

PC disk includes POPOUT-LITE, the powerful stereogram generator, plus sample images for you to play with!

Includes a full-color insert of sample RDS images!

$24.99 USA / $33.99 CAN
£23.49 Net UK (incl. of VAT)

ISBN 1-56529-994-9

90000

9 781565 299948

que